Oracle® Warehouse Builder

Case Book

10*g* Release 1 (10.1)

Part No. TBD-01

July 2004

ORACLE®

Oracle Warehouse Builder Case Book 10*g* Release 1 (10.1)

Part No. TBD-01

Copyright © 2001, 2003, Oracle. All rights reserved.

Primary Author: Padmaja Potineni, Shirinne Alison, Kavita Nayar, Michelle Bird, Julia Stein

Contributing Author: Igor Machin, Jean-Pierre Dijcks, Mark Van de Wiel, Nikolai Rochnik, Michael Bethke, Edwin Meijer

The Programs (which include both the software and documentation) contain proprietary information; they are provided under a license agreement containing restrictions on use and disclosure and are also protected by copyright, patent, and other intellectual and industrial property laws. Reverse engineering, disassembly, or decompilation of the Programs, except to the extent required to obtain interoperability with other independently created software or as specified by law, is prohibited.

The information contained in this document is subject to change without notice. If you find any problems in the documentation, please report them to us in writing. This document is not warranted to be error-free. Except as may be expressly permitted in your license agreement for these Programs, no part of these Programs may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose.

If the Programs are delivered to the United States Government or anyone licensing or using the Programs on behalf of the United States Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are "commercial computer software" or "commercial technical data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the Programs, including documentation and technical data, shall be subject to the licensing restrictions set forth in the applicable Oracle license agreement, and, to the extent applicable, the additional rights set forth in FAR 52.227-19, Commercial Computer Software--Restricted Rights (June 1987). Oracle Corporation, 500 Oracle Parkway, Redwood City, CA 94065

The Programs are not intended for use in any nuclear, aviation, mass transit, medical, or other inherently dangerous applications. It shall be the licensee's responsibility to take all appropriate fail-safe, backup, redundancy and other measures to ensure the safe use of such applications if the Programs are used for such purposes, and we disclaim liability for any damages caused by such use of the Programs.

Oracle is a registered trademark of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

The Programs may provide links to Web sites and access to content, products, and services from third parties. Oracle is not responsible for the availability of, or any content provided on, third-party Web sites. You bear all risks associated with the use of such content. If you choose to purchase any products or services from a third party, the relationship is directly between you and the third party. Oracle is not responsible for: (a) the quality of third-party products or services; or (b) fulfilling any of the terms of the agreement with the third party, including delivery of products or services and warranty obligations related to purchased products or services. Oracle is not responsible for any loss or damage of any sort that you may incur from dealing with any third party.

Contents

Send Us Your Comments v

Preface vii

1 What is the Fastest Way to Load Data from Flat Files? 1-1

2 How Do I Transfer Remote Files? 2-1

3 How Do I Load Data Stored in a Microsoft Excel File?..... 3-1

4 How Do I Import Designs from Third-Party Tools Such as CA ERwin? 4-1

5 How Do I Load Transaction Data?..... 5-1

6 How Do I Handle Existing PL/SQL Code with Warehouse Builder? 6-1

7 How Do I Handle Flat Files with Variable Names? 7-1

8 How Do I Update the Target Schema?..... 8-1

9 How Do I Manage Multiple Versions of my BI Implementation?..... 9-1

10 How Do I Inspect Error Logs in Warehouse Builder? 10-1

11 How Do I Export Metadata to Oracle Discoverer? 11-1

12 How Do I Export a Warehouse Builder Design to a File? 12-1

Index

Send Us Your Comments

Oracle Warehouse Builder Case Book 10*g* Release 1 (10.1)

Part No. TBD-01

Oracle welcomes your comments and suggestions on the quality and usefulness of this publication. Your input is an important part of the information used for revision.

- Did you find any errors?
- Is the information clearly presented?
- Do you need more information? If so, where?
- Are the examples correct? Do you need more examples?
- What features did you like most about this manual?

If you find any errors or have any other suggestions for improvement, please indicate the title and part number of the documentation and the chapter, section, and page number (if available). You can send comments to us in the following ways:

- Electronic mail: **dwhdoc_us@oracle.com**
- Postal service:

  Oracle Corporation
  Oracle Warehouse Builder Documentation
  500 Oracle Parkway, Mailstop 2op10
  Redwood Shores, CA 94065
  U.S. A.

If you would like a reply, please give your name, address, telephone number, and electronic mail address.

If you have problems with the software, please contact your local Oracle Support Services.

Preface

This preface includes the following topics:

- Purpose on page vii
- Audience on page vii
- Documentation Accessibility on page viii
- How This Book Is Organized on page viii
- Conventions on page ix
- Related Publications on page ix
- Contacting Oracle on page x

Purpose

Oracle Warehouse Builder is a comprehensive toolset for practitioners who move and transform data, develop and implement business intelligence systems, perform metadata management, or create and manage Oracle databases and metadata.

The *Oracle Warehouse Builder Case Book* addresses some common business scenarios that await companies implementing a Business Intelligence system. Unlike the *Oracle Warehouse Builder User's Guide*, this book does not give click-by-click instructions for Warehouse Builder features. Instead, this book follows a fictional company, and, using a common set of example data, addresses questions that start with "What if we need to...?" and "What is the best way to...?".

Audience

This book is intended for:

- Business Intelligence application developers
- Warehouse architects, designers, and developers—especially SQL and PL/SQL developers
- Data analysts and those who develop extract, transform, and load routines
- Developers of large-scale products based on data warehouses
- Warehouse administrators
- System administrators
- Other MIS professionals

To use the information in this book, you need to be comfortable with the concepts of Relational Database Management Systems and Data Warehouse design. For information on data warehousing, refer to the *Oracle Data Warehousing Guide*. Also, you need to be familiar with Oracle's relational database software products such as Oracle Database, SQL*Plus, SQL*Loader, Oracle Enterprise Manager, and Oracle Workflow. Consult the *Oracle Warehouse Builder User's Guide* for specific instructions on functionality to which the case studies in this book refer.

Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible, with good usability, to the disabled community. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Standards will continue to evolve over time, and Oracle is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For additional information, visit the Oracle Accessibility Program Web site at

`http://www.oracle.com/accessibility/`

Accessibility of Code Examples in Documentation JAWS, a Windows screen reader, may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, JAWS may not always read a line of text that consists solely of a bracket or brace.

Accessibility of Links to External Web Sites in Documentation This documentation may contain links to Web sites of other companies or organizations that Oracle does not own or control. Oracle neither evaluates nor makes any representations regarding the accessibility of these Web sites.

How This Book Is Organized

The *Oracle Warehouse Builder Case Book* contains case studies that illustrate common Warehouse Builder functionality. The case studies are organized into the following chapters:

- Chapter 1, "What is the Fastest Way to Load Data from Flat Files?"
- Chapter 3, "How Do I Load Data Stored in a Microsoft Excel File?"
- Chapter 4, "How Do I Import Designs from Third-Party Tools Such as CA ERwin?"
- Chapter 5, "How Do I Load Transaction Data?"
- Chapter 6, "How Do I Handle Existing PL/SQL Code with Warehouse Builder?"
- Chapter 2, "How Do I Transfer Remote Files?"
- Chapter 7, "How Do I Handle Flat Files with Variable Names?"
- Chapter 8, "How Do I Update the Target Schema?"
- Chapter 9, "How Do I Manage Multiple Versions of my BI Implementation?"
- Chapter 10, "How Do I Inspect Error Logs in Warehouse Builder?"

Conventions

In this manual, Windows refers to the Windows NT, Windows 2000, and Windows XP operating systems. The SQL*Plus interface to Oracle Database may be referred to as SQL.

In the examples, an implied carriage return occurs at the end of each line, unless otherwise noted. You must press the Return key at the end of a line of input.

The following table lists the conventions used in this manual:

Convention	Meaning
. . .	Vertical ellipsis points in an example mean that information not directly related to the example has been omitted.
...	Horizontal ellipsis points in statements or commands mean that parts of the statement or command not directly related to the example have been omitted.
boldface text	Boldface type in text refers to interface buttons and links. Boldface type also serves as emphasis to set apart main ideas.
italicized text	Italicized text applies to new terms introduced for the first time. Italicized text also serves as an emphasis on key concepts.
`unicode text`	Unicode text denotes exact code, file directories and names, and literal commands.
`italicized unicode text`	Italicized unicode text refers to parameters whose value is specified by the user.
[]	Brackets enclose optional clauses from which you can choose one or none.

Related Publications

The Warehouse Builder documentation set includes these manuals:

- *Oracle Warehouse Builder Case Book*
- *Oracle Warehouse Builder User's Guide*
- *Oracle Warehouse Builder Installation and Configuration Guide*
- *Oracle Warehouse Builder Transformation Guide*
- *Oracle Warehouse Builder Scripting Reference*
- *Oracle Warehouse Builder Release Notes*

In addition to the Warehouse Builder documentation, you can refer to the *Oracle Data Warehousing Guide.*

Oracle provides additional information sources, including other documentation, training, and support services that can enhance your understanding and knowledge of Oracle Warehouse Builder .

- For more information on Oracle Warehouse Builder technical support, contact Oracle World Wide Support services at:

  `http://www.oracle.com/support`

- For the latest information on, and downloads of, software and documentation updates to Oracle Warehouse Builder , visit Oracle*MetaLink* at:

http://metalink.oracle.com

- You can order other Oracle documentation at:

  http://oraclestore.oracle.com

Contacting Oracle

Oracle*MetaLink*

Oracle*MetaLink* is the Oracle support Web site where you can find the latest product information, including documentation, patch information, BUG reports, and TAR entries. Once registered, you can access email, phone, and Web resources for all Oracle products. *MetaLink* is located at:

http://metalink.oracle.com

Check *MetaLink* regularly for Warehouse Builder information and updates.

Documentation

You can order Oracle product documentation by phone or through the World Wide Web:

- **Phone:** Call 800-252-0303 to order documentation or request a fax listing of available Oracle documentation.
- **Oracle Documentation Sales Web site:** http://oraclestore.oracle.com
- **Oracle Support Services Web site:** http://www.oracle.com/support

1

What is the Fastest Way to Load Data from Flat Files?

Scenario

The weekly sales data of a company is stored in a flat file called `weeklysales.txt`. This data needs to be loaded into a table in the Warehouse Builder repository.

An example data set of the source file is defined as:

```
SALESREP, MONTH,PRODUCT_ID,W1_QTY,W2_QTY,W3_QTY.W4_QTY
100,JAN02,3247,4,36,21,42
101,JUL02,3248,24,26,4,13
```

Each record in the file contains details of the quantity sold by each sales representative in each week of a month. This data needs to be loaded into the Warehouse Builder repository.

Solution

Warehouse Builder provides two methods of loading data stored in flat files into the Design Repository. The methods are:

- Solution 1: Using SQL*Loader
- Solution 2: Using External Tables

Table 1–1 Difference Between SQL*Loader and External Tables

SQL*Loader	External tables
Transformation and loading of data are two separate steps. Data must be loaded into a staging area and then transformed.	Combines the transformation and loading of data within a single SQL DML statement. There is no need to stage the data temporarily before inserting it into the target table.

SQL *Loader

SQL*Loader is an Oracle tool that enables you to load data from flat files into tables in an Oracle database. In Warehouse Builder, use the Flat File operator to load data using SQL*Loader.

SQL*Loader is the only method you can use to load data from a flat file into a repository whose database version is Oracle8*i* Release 3 (8.1.7) or earlier.

When To Use SQL*Loader

Use SQL*Loader to load data from a flat file if:

- The database version is Oracle8*i* Release 3 (8.1.7) or earlier.
- No complex transformations are required on the input data.

External Tables

An external table is a database object that enables you to access data stored in external sources. External tables allow flat files to have the same properties as database tables (read-only) and extend the power of SQL to reading flat files. You can also query, join, transform, and constrain the flat file data before loading it into the database.

Note: External tables are supported only from Oracle9*i* onwards.

In Warehouse Builder, use the external table object and the Mapping External Table operator to load data from a flat file into the repository. The design benefit of using external tables is that it extends additional database features to a flat file. By using external tables instead of flat files, you can apply complex transformations to the data in flat files that were previously only used for relational files.

Benefits of Using External Tables

- Provides faster access to flat files because the external data can be accessed in parallel during a load.
- Can perform heterogeneous joins with database tables or other external tables.

When To Use External Tables

- To transform flat file data before loading into the database
- To perform complex transformations, such as joins and aggregations, on the flat file data before loading it into the Warehouse Builder repository

External tables can be faster when the following conditions are met:

- The hardware has multiple processors.
- The flat file is large (has many records).

When these conditions are met, the benefits of parallel processing will outperform SQL*Loader processing.

Solution 1: Using SQL*Loader

Use SQL*Loader to load data from the flat file into the target table. Warehouse Builder provides the Flat File operator that enables you to load data into a target table using SQL*Loader.

However, the transformations that you can perform on data loaded using a flat file operator are limited to SQL*Loader transformations only. You can use only the following mapping operators when you use a flat file operator as a source:

- Filter operator
- Constant operator
- Data Generator operator
- Mapping Sequence operator

- Expression operator
- Transformation operator

To load data using SQL*Loader, create a mapping that uses the mapping flat file operator to represent the source data. Map the output of this operator directly to the target table.

Solution 2: Using External Tables

Use external tables to load data from the flat file `weeklysales.txt` into the repository table `SALES_DATA`. Create a mapping that contains the External Table operator as the source. This External Table operator must be bound to the external table object that you create referring to the flat file. Map the output of the external table operator directly to the target table.

Mapping to Load Data Using External Tables

In the mapping that loads the data from the flat file, use the External Table operator to represent the source data. Map the output of the External Table operator to the target table `SALES_DATA`.

2

How Do I Transfer Remote Files?

Scenario

Developers at your company designed mappings that extract, transform, and load data. The source data for the mapping resides on a server separate from the server that performs the ETL processing. You would like to create logic that transfers the files from the remote machine and triggers the dependent mappings.

Solution

In Warehouse Builder, you can design a process flow that executes file transfer protocol (FTP) commands and then launches a mapping. For the process flow to be valid, the FTP commands must involve transferring data either from or to the server with the Runtime Service installed. To move data between two machines, neither of which host the Runtime Service, first transfer the data to the Runtime Service host machine and then transfer the data to the second machine.

You can design the process flow to launch different activities depending upon the success or failure of the FTP commands.

Case Study

This case study describes how to transfer files from one machine to another and launch a dependent mapping. The case study provides examples of all the necessary servers, files, and user accounts.

- **Data host machine:** For the machine hosting the source data, you need a user name and password, host name, and the directory containing the data. In this case study, the machine hosting the data is a UNIX server named `salessrv1`. The source data is a flat file named `salesdata.txt` located in the `/usr/stage` directory.
- **Runtime Service host machine:** In this case study, Warehouse Builder and the Runtime Service are installed on a machine called `local` with a Windows operating system. `Local` executes the mapping and the process flow.
- **Mapping:** This case study assumes there is a mapping called `salesresults` that uses a copy of `salesdata.txt` stored on `local` at `c:\temp` as its source.
- **FTP Commands:** This case study illustrates the use of a few basic FTP commands on the Windows operating system.

Your objective is to create logic that ensures the flat file on `salessrv1` is copied to the `local` machine and then trigger the execution of the `salesresults` mapping.

To transfer files and launch a dependent mapping, refer to the following sections:

1. "Defining Locations" on page 2-2 for details.

2. "Creating the Process Flow" on page 2-2
3. "Setting Parameters for the FTP Activity" on page 2-2
4. "Configuring the FTP Activity" on page 2-5
5. "Registering the Process Flow for Deployment" on page 2-6

After you complete the instructions in the above sections, you can run the process flow.

Defining Locations

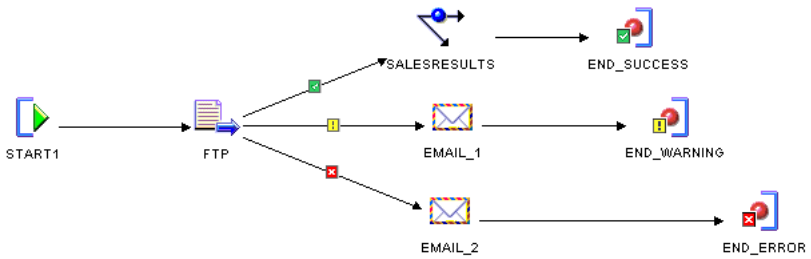
Locations are logical representations of the various data sources and destinations in the warehouse environment. In this scenario, the locations are the logical representations of the host and path name information required to access a flat file. Warehouse Builder requires these definitions for deploying and running the process flow. When you deploy the process flow, Warehouse Builder prompts you to type the host and path name information associated with each location. You must define locations for each machine involved in the data transfer.

To define locations, right-click the appropriate Locations node on the navigation tree and right-click or press **Ctrl+N** to create a new location. For `salessrv1`, right-click **File Location** and create a location named `REMOTE_FILES`. Repeat the step for `local` and create the location `LOCAL_FILES`.

Creating the Process Flow

Use the Process Flow Editor to create a process flow with an FTP activity that transitions to the `salesresults` mapping on the condition of success. Your process flow should appear similar to Figure 2–1.

Figure 2–1 Process Flow with FTP Transitioning to a Mapping



Setting Parameters for the FTP Activity

This section describes how to specify the commands for transferring data from the remote server, salessrv1, to the `local` machine. You specify the FTP parameters by typing values for the FTP activity parameters on the Activity View as displayed in Figure 2–2.

Warehouse Builder offers you flexibility on how you specify the FTP commands. Choose one of the following methods:

- **Method 1: Write a script in Warehouse Builder:** Choose this method when you want to maintain the script in Warehouse Builder and/or when password security to servers is a requirement.

For this method, write or copy and paste the script into the VALUE column of the SCRIPT parameter. In the COMMAND parameter, type the path to the FTP executable such as `c:\winnt\system32\ftp.exe`. Also, type the `Task.Input` variable into the VALUE column of the PARAMETER_LIST parameter.

- **Method 2: Call a script maintained outside of Warehouse Builder:** If password security is not an issue, you can direct Warehouse Builder to a file containing a script including the FTP commands and the user name and password.

  To call a file on the file system, type the appropriate command in PARAMETERS_LIST to direct Warehouse Builder to the file. For a Windows operating system, type the following: `?"-s:<file path\file name>"?`

  For example, to call a file named `move.ftp` located in a temp directory on the C drive, type the following: `?"-s:c:\temp\move.ftp"?`

  Leave the SCRIPT parameter blank for this method.

Example: Writing a Script in Warehouse Builder for the FTP Activity

The following example illustrates Method 1 described above. It relies on a script and the use of substitution variables. The script navigates to the correct directory on `salessrv1` and the substitution variables are used for security and convenience.

This example assumes a Windows operating system. For other operating systems, issue the appropriate equivalent commands.

To define a script within the FTP activity:

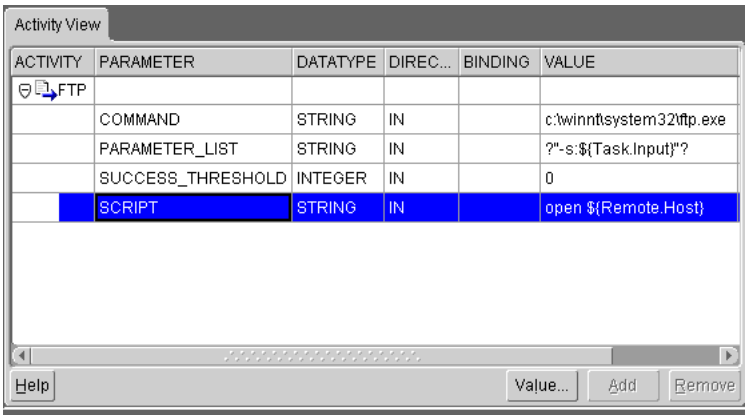
1. Select the FTP activity on the canvas to view and edit activity parameters in the Activity View displayed in the lower left panel of the Process Flow Editor.
2. For the **COMMAND** parameter, type the path to the FTP executable in the column labelled `Value`. If necessary, use the scroll bar to scroll to the right and reveal the column labelled `Value`.

   For windows operating systems, the FTP executable is often stored at `c:\winnt\system32\ftp.exe`.
3. For **PARAMETER_LIST**, type the `Task.Input` variable.

   When defining a script in Warehouse Builder and using Windows FTP, you must type `?"-s:${Task.Input}"?` into PARAMETER_LIST.

   For UNIX, type `?"${Task.Input}"?`.
4. Navigate and highlight the **SCRIPT** parameter. Your Activity View should display similar to Figure 2–2.

Figure 2–2 Activity View for FTP Activity Using a Script



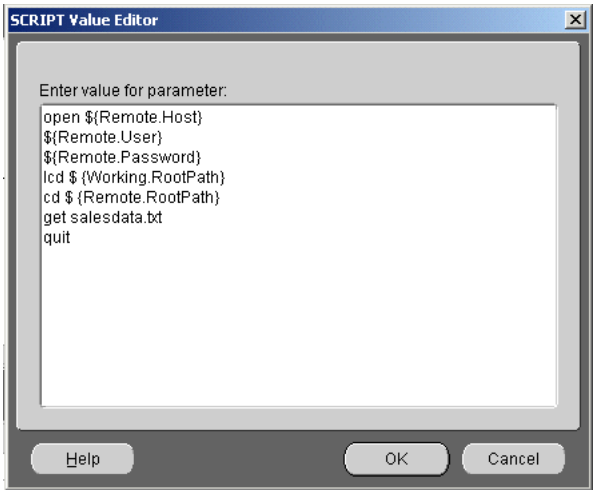
5. Click **Value...** displayed at the bottom of the Activity View.

   Warehouse Builder displays the SCRIPT Value Editor. Write or copy and paste FTP commands into the editor.

   Figure 2–2 shows a script that opens a connection to the remote host, changes the directory to the local machine, changes the directory to the remote host, transfers the file, and closes the connection.

   Notice that the script in Figure 2–2 includes `${Remote.User}` and `${Remote.Password}`. These are substitution variables. Refer to "Using Substitution Variables" for more details.

Figure 2–3 SCRIPT Value Editor Using Substitution Variables



Using Substitution Variables

Substitution variables are available only when you choose to write and store the FTP script in Warehouse Builder.

Use substitution variables to prevent having to update FTP activities when server files, accounts, and passwords change. For example, consider that you create 10 process flows that utilize FTP activities to access a file on `salessrv1` under a specific directory. If the file is moved, without the use of substitution variables, you must update each FTP activity individually. With the use of substitution variables, you need only update the location information as described in "Defining Locations" on page 2-2.

Substitution variables are also important for maintaining password security. When Warehouse Builder executes an FTP activity with substitution variables for the server passwords, it resolves the variable to the secure password you provided for the associated location.

Table 2–1 lists the substitute variables you can provide for the FTP activity. `Working` refers to the machine hosting the Runtime Service, the *local* machine in this case study. `Remote` refers to the other server involved in the data transfer. You designate which server is remote and local when you configure the FTP activity. For more information, see "Configuring the FTP Activity" on page 2-5.

Table 2–1 Substitute Variables for the FTP Activity

Variable	Value
${Working.RootPath}	The root path value for the location of the Runtime Service host.
${Remote.Host}	The host value for the location involved in transferring data to or from the Runtime Service host.
${Remote.User}	The user value for the location involved in transferring data to or from the Runtime Service host.
${Remote.Password}	The password value for the location involved in transferring data to or from the Runtime Service host.
${Remote.RootPath}	The root path value for the location involved in transferring data to or from the Runtime Service host.

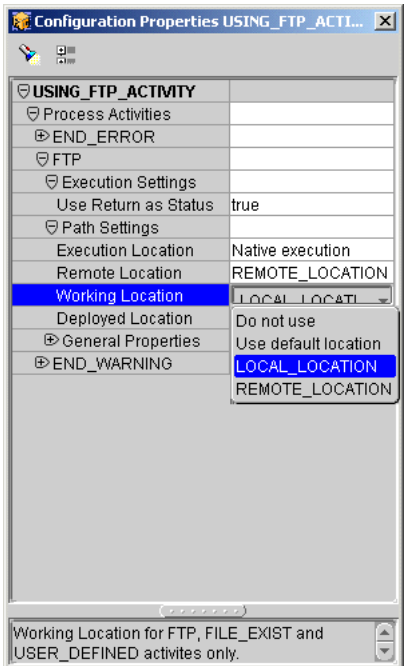
Configuring the FTP Activity

As part of configuring the complete process flow, configure the FTP activity.

To configure the FTP Activity:

1. Right-click the process flow on the navigation tree and select **Configure.**
2. Expand the FTP activity and the Path Settings. Warehouse Builder displays the configuration settings as shown in Figure 2–4.

Figure 2–4 Configuration Settings for the FTP Activity



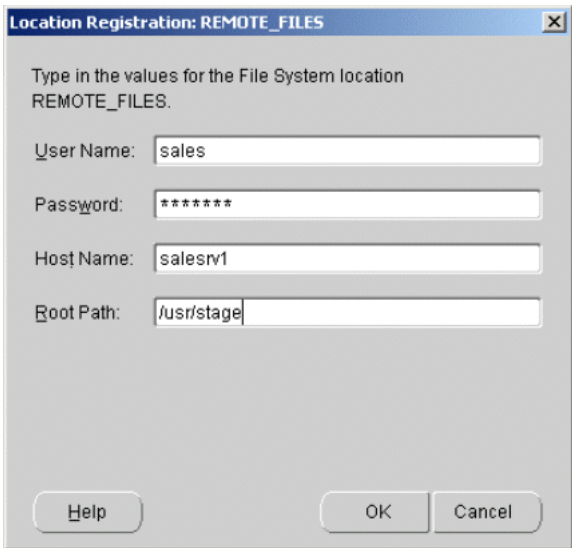
3. Set Remote Location to `REMOTE_LOCATION` and Working Location to `LOCAL_LOCATION`.
4. Set **Use Return as Status** to true. This ensures that the process flow uses the FTP return codes for determining which outgoing transition to activate. For the process flow in this case study, shown in Figure 2–1 on page 2-2, if FTP returns a success value of 1, the process flow continues down the success transition and executes the `salesresults` mapping.

Registering the Process Flow for Deployment

After you complete these instructions, you can deploy and run the process flow. To deploy the process flow, launch the Deployment Manager by right-clicking and selecting **Deploy** from either the process flow module or package on the navigation tree. The Deployment Manager prompts you to register the `REMOTE_LOCATION` and the `LOCAL_LOCATION`.

Figure 2–5 shows the registration information for the `REMOTE_LOCATION`. For the `LOCAL_FILES`, only the root path is required.

Figure 2–5 Example Location Registration Information



Now you can run the process flow.

3

How Do I Load Data Stored in a Microsoft Excel File?

Scenario

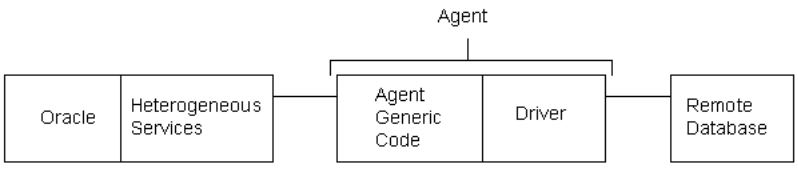
A company stores its employee data in an Excel file called `employees.xls`. This file contains two worksheets: `employee_details` and `job_history`. You need to load the data from the `employee_details` worksheet into a target table in Warehouse Builder.

Solution

To load data stored in an Excel file into a target table, you must first use the Excel file as a source. Warehouse Builder enables you to source data stored in a non-Oracle source, such as Microsoft Excel, using the Heterogeneous Services component of the Oracle database.

Figure 3–1 describes how the Oracle database uses Heterogeneous services to access a remote non-Oracle source.

Figure 3–1 Heterogeneous Services Architecture



The Heterogeneous Services component in the database communicates with the Heterogeneous Services agent process. The agent process, in turn, communicates with the remote database.

The agent process consists of agent-generic code and a system-specific driver. All agents contain the same agent-generic code. But each agent has a different driver depending on the type of data being sourced.

Case Study

This case study shows you how to use an Excel file called `employees.xls` as a source in Warehouse Builder.

Step 1: Install ODBC Driver for Excel

To read data from Microsoft Excel, you must have the ODBC driver for Excel installed.

Step 3: Prepare the Excel File (Optional)

To source data from an Excel file, define a name for the range of data being sourced:

1. In the `employee_details` worksheet, highlight the range that you want to query from Oracle.

   The range should include the column names and the data. Ensure that the column names confirm to the rules for naming columns in the Oracle database.

2. From the **Insert** menu, select **Name** and then **Define.** The **Define Name** dialog is displayed. Specify a name for the range.

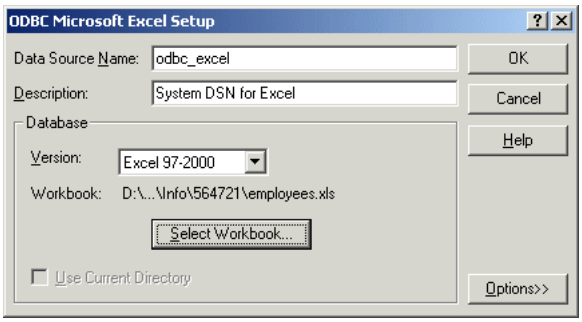
Step 3: Create a System DSN

Set up a System Data Source Name (DSN) using the Microsoft ODBC Administrator.

1. Select **Start**, followed by **Programs**, ***DATABASE_ORACLE_HOME***, **Network Administration**, and then **Microsoft ODBC Administrator.**
2. Navigate to the **System DSN** tab and click **Add** to create a System DSN.
3. Select **Microsoft Excel** as the driver for which you want to set up the data source.

   The ODBC Microsoft Excel Setup dialog displays as shown in Figure 3–2.

Figure 3–2 ODBC Microsoft Excel Setup Dialog



4. Specify the name of the DSN as `odbc_excel`.
5. Click **Select Workbook** to select the Excel file from which you want to read the data.
6. Verify that the Version field lists the version of the source Excel file accurately.

Step 4: Create the Heterogeneous Services Initialization File

To configure the agent, you must set the initialization parameters in the heterogeneous services initialization file. Each agent has its own heterogeneous services initialization file. The name of the Heterogeneous Services initialization file is `init`*`SID`*`.ora`, where *`SID`* is the Oracle system identifier used for the agent. This file is located in the *`DATABASE_ORACLE_HOME`*`/hs/admin` directory.

Create the `initExcel_SID.ora` file in the *`DATABASE_ORACLE_HOME`*`/hs/admin` directory as follows:

```
HS_FDS_CONNECT_INFO = odbc_excel
HS_AUTOREGISTER = TRUE
HS_DB_NAME = hsodbc
```

Here, *odbc_excel* is the name of the system DSN you created in Step 3. Excel_SID is the name of the Oracle system identifier used for the agent.

Step 5: Modify the listener.ora file

Set up the listener on the agent to listen for incoming requests from the Oracle Database server. When a request is received, the agent spawns a Heterogeneous Services agent. To set up the listener, modify the entries in the listener.ora file located in the *DATABASE_ORACLE_HOME*/network/admin directory as follows:

```
SID_LIST_LISTENER =
  (SID_LIST =
    (SID_DESC =
      (SID_NAME = Excel_SID)
      (ORACLE_HOME = c:\oracle\db92)
      (PROGRAM = hsodbc)
    )
    (SID_DESC =
      (SID_NAME = PLSExtProc)
      (ORACLE_HOME = c:\oracle\db92)
      (PROGRAM = extproc)
      )
  )
```

1. For the SID_NAME parameter, use the SID that you specified in Step 4 when creating the initialization parameter file for the Heterogeneous Services.
2. Ensure that the ORACLE_HOME parameter value is the path to your Oracle Home directory.
3. The value associated with the PROGRAM keyword defines the name of the agent executable.

Remember to restart the listener after making these modifications.

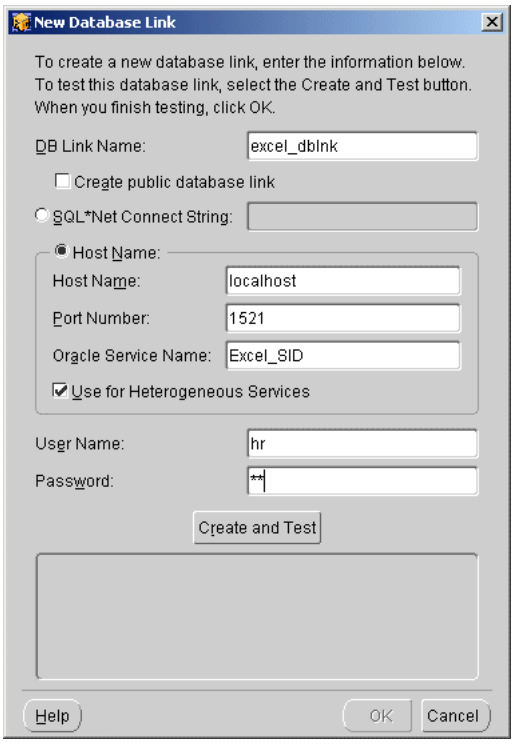
Note: Ensure that the initialization parameter GLOBAL_NAMES is set to FALSE in the database's initialization parameter file. FALSE is the default setting for this parameter.

Step 6: Create an ODBC Source Module and a Database Link

Use the following steps to create an ODBC source module and database link:

1. From the Warehouse Builder console, create an ODBC source module. On the navigation tree, ODBC modules are listed under the Others node of the Databases node.
2. On the **Connection Information** page, click **New Database Link** to create a new database link that reads data using the data source created. Figure 3–3 shows the entries used on the New Database Link dialog.

Figure 3–3 New Database Link Dialog



Notice that the **Oracle Service Name** field uses the Oracle system identifier specified for the agent.

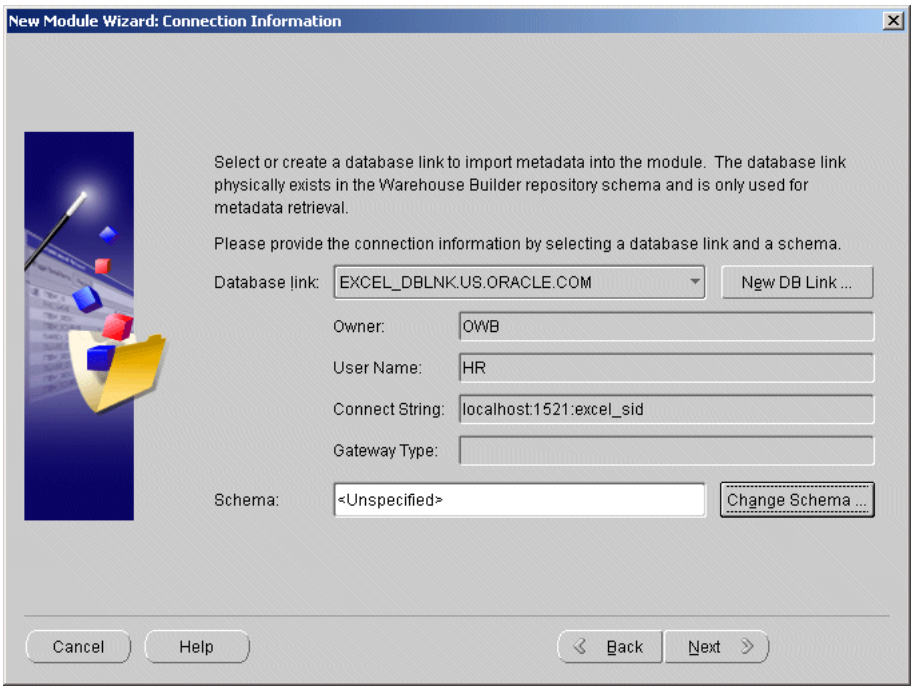
3. Ensure that the **Use for Heterogeneous Services** option is selected.

   Because you are not accessing an Oracle database, you can enter any value for username and password.

4. Create and test this database link. Close the **New Database Link** dialog.

5. Leave the **Schema** name **<unspecified>**. Click the **Change Schema** button and select **<unspecified>**. The **Connection Information** page now looks as shown in Figure 3–4.

Figure 3–4 Connection Information Page



6. Create a new deployment location for the module or specify an existing location.

Step 7: Import Metadata from Excel Using the Metadata Import Wizard

Use the Metadata Import wizard to import metadata from the Excel file into Warehouse Builder. Select Tables as the Filter condition. The wizard displays all the worksheets in the source Excel file under the Tables node in the list of available objects.

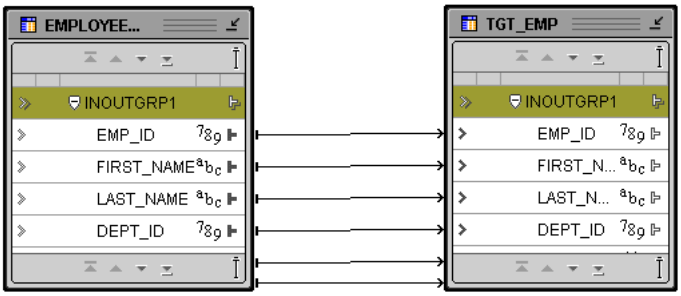
1. Select `employee_details` and use the arrow to move it to the list of selected objects.
2. Click Finish to import the data.

   The data from the employee_details worksheet is now stored in a table called `employee_details` in the ODBC source module created in Step 6.

Step 8: Create a Mapping to Load Data Into the Target Table

In the Warehouse Builder console, expand the module that contains the target table. Use the table called `employee_details` in the ODBC source module as a source to load data into the target table. Figure 3–5 displays the mapping used to load data into the target table.

Figure 3–5 Mapping to Load Data Into the Target Table



Step 9: Deploy the Mapping

Use the Deployment Manager to deploy the mapping you created in Step 8. Ensure that you first deploy the source module before you deploy the mapping.

4

How Do I Import Designs from Third-Party Tools Such as CA ERwin?

Scenario

A movie rental company uses tools from different vendors for data modelling, extraction, transformation and loading (ETL), and reporting purposes. Using a variety of tools has led to several metadata integration issues for this company. Often, the design work done using one tool cannot be completely integrated or reused in another. This company wants to find a method to streamline and integrate all its metadata designs and ETL processes using a single tool.

Solution

Warehouse Builder enables the company to import and integrate metadata designs from different tools and use them for data modelling and ETL purposes using only one tool. Warehouse Builder uses the seamlessly integrated technology from its partner, Meta Integration Technology Inc. (MITI), to import the metadata and reuse the data models designed by other third party tools.

This case study shows you how to easily import design files developed using CA ERwin into Warehouse Builder. You can then reuse the metadata for ETL design and reporting using a single tool. You can follow this model to import files from other tools such as Sybase PowerDesigner, Business Objects Designer, etc.

Case Study

This case study shows you how the movie rental company can migrate their ERwin data model designs into Warehouse Builder. They can also use this model to import designs from other third party tools and consolidate their design metadata in a central repository. Follow these steps:

- Download metadata from CA ERwin.
- Install MITI on your local system after purchasing a license for this product.
- Use Warehouse Builder Transfer Wizard to import the ERwin metadata into Warehouse Builder.

Step1: Download metadata from CA ERwin Download the design metadata from CA ERwin to your local system.

Step 2: Install the Meta Integration Model Bridge Warehouse Builder enables you to integrate with Meta Integration Model Bridges (MIMB). These bridges translate metadata from a proprietary metadata file or repository to the standard CWM format that can be imported into Warehouse Builder using the Warehouse Builder Transfer Wizard. To import files from different design tools into Warehouse Builder, you must

first obtain an MIMB license and install the bridges on your system. Follow these steps to complete the installation.

To download MIMB:

1. Download the Model Bridge product from the following web site:

   http://www.metaintegration.net/Products/Downloads/

2. Install the MIMB by running the setup on your system.

3. During installation, choose **Typical with Java Extensions** as the installation type from the Setup Type page.

   If the set up program is not able to find a JDK on your machine, you must provide the JNI library directory path name. Your path environment variable must contain the metaintegration directory. If not, you need to add it to the path:

   ```
   c:\program files\metaintegration\win32
   ```

Step 3: Import Metadata from CA ERwin using Warehouse Builder After you install the MIMB product, follow these steps to import metadata from ERwin and other third party design tools:

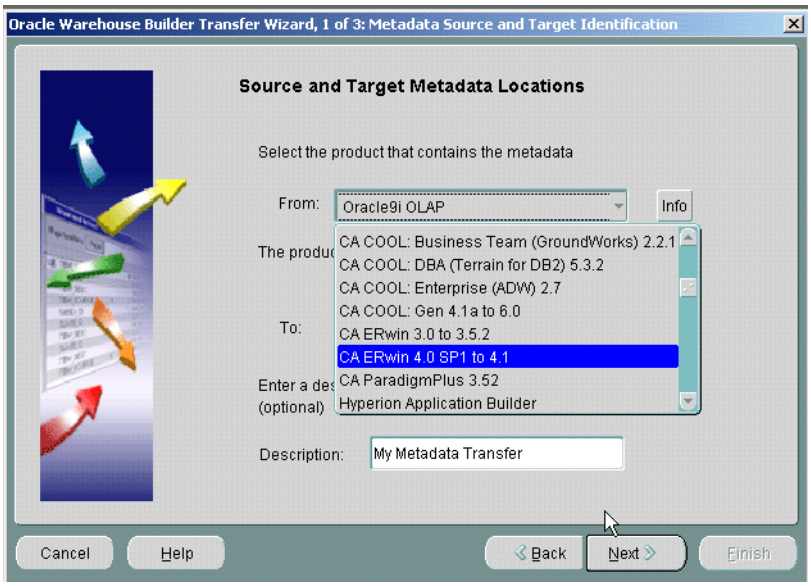
1. From the Warehouse Builder console, select and expand the Project node to which you want to import the metadata. In this example, the ERwin files are imported into MY_PROJECT.

2. From the **Project** menu, select **MetaData Import,** and then **Bridge** to start the **Warehouse Builder** Transfer Wizard.

   The Transfer Wizard seamlessly integrates with the MITI technology to translate the third party metadata into a standard CWM format that is imported into Warehouse Builder. Follow the wizard to complete the import.

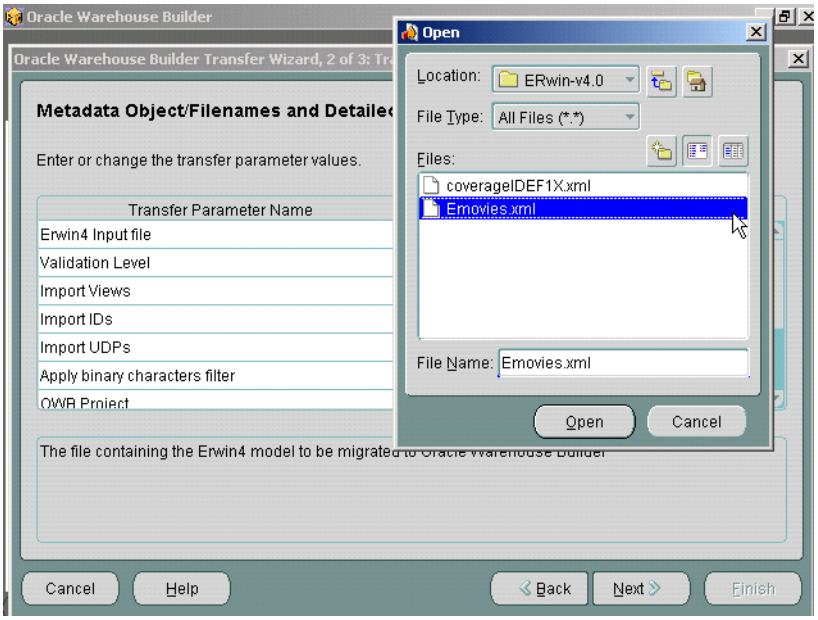
3. In the Metadata Source and Target Identification page, select the **CA ERwin 4.0 SP1 to 4.1** option in the **From** field as shown in Figure 4–1.

Figure 4–1 Transfer Wizard Source and Target Identification Page



4. In the Transfer Parameter Identification page, provide the path where the ERwin files are located in the **Erwin4 Input File** field. In this example, the company wants to import the `Emovies.xml` file from ERwin, as shown in Figure 4–2.

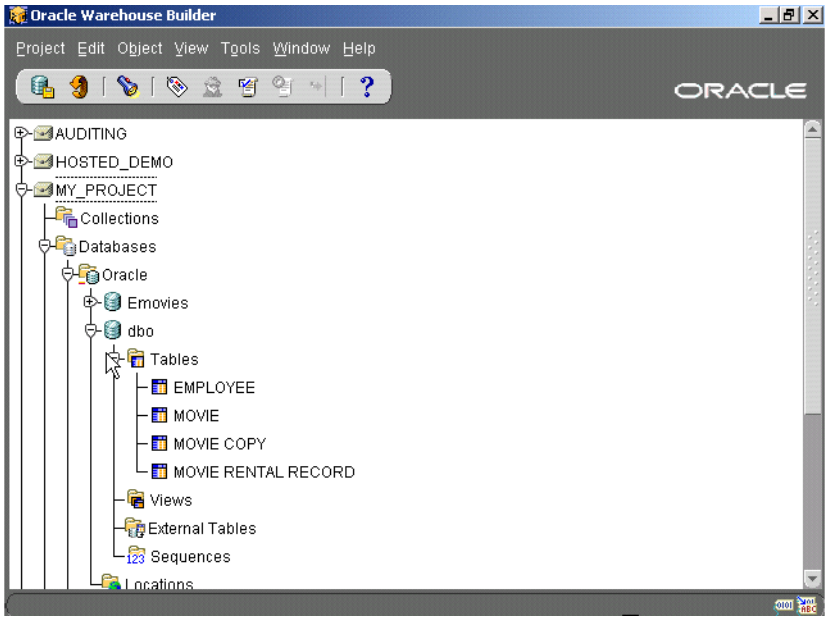
Figure 4–2 Transfer Parameter Identification Page



5. Accept the default options for all other fields. In the OWB Project field, select the Warehouse Builder project where you want to import the ERwin file.
6. Complete the remaining wizard steps and finish the import process.

Step 4: Verify the Imported Objects After you finish importing the ERwin files into Warehouse Builder, expand the MY_PROJECT folder, then the Databases node, and then the Oracle node. You can see the imported source metadata objects in the repository, as shown in Figure 4–3.

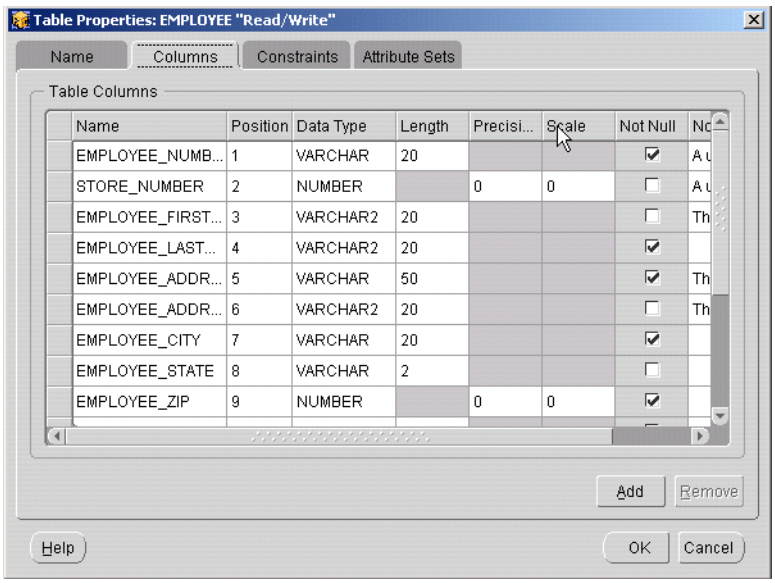
Figure 4–3 Metadata Objects Imported from CA Erwin



7. In this example, two source modules: `Emovies` and `dbo` have been imported into Warehouse Builder. The dbo source module contains four tables: EMPLOYEE, MOVIE, MOVIE COPY, and MOVIE RENTAL RECORD.

Double-click the table names to see the properties for each of these tables. Warehouse Builder imports all the metadata including descriptions and detailed information on table columns and constraints, as shown in Figure 4–4.

Figure 4–4 Table Properties Imported from CA Erwin



The designers at the movie rental company can use these sources tables to model ETL designs in Warehouse Builder, generate ETL code, and run reports on them. Further, Warehouse Builder enables them to easily import all the scattered third-party design metadata and consolidate all their design and development efforts in the Warehouse Builder design repository.

5

How Do I Load Transaction Data?

Scenario

Your company records all its transactions as they occur, resulting in inserts, updates, and deletes, in a flat file called `record.csv`. These transactions need to be processed in the exact order they were stored. For example, if an order was first placed, then updated, then cancelled and re-entered, this transaction must be processed exactly in the same order.

An example data set of the source file `record.csv` is defined as:

Action,DateTime,Key,Name,Desc

```
I,71520031200,ABC,ProdABC,Product ABC
I,71520031201,CDE,ProdCDE,Product CDE
I,71520031202,XYZ,ProdXYZ,Product XYZ
U,71620031200,ABC,ProdABC,Product ABC with option
D,71620032300,ABC,ProdABC,Product ABC with option
I,71720031200,ABC,ProdABC,Former ProdABC reintroduced
U,71720031201,XYZ,ProdXYZ,Rename XYZ
```

You want to load the data into a target table such as the following:

```
SRC_TIMESTA KEY NAME    DESCRIPTION
----------- --- ------- --------------------------
71520031201 CDE ProdCDE Product CDE
71720031201 XYZ ProdXYZ Rename XYZ
71720031200 ABC ProdABC Former ProdABC reintroduced
```

Solution

Warehouse Builder enables you to design ETL logic and load the data in the exact temporal order in which the transactions were stored at source. To achieve this result, you design a mapping that orders and conditionally splits the data before loading it into the target. Then, you configure the mapping to generate code in row based operating mode. In row based operating mode Warehouse Builder generates code to process the data row by row using if-then-else constructions, as shown in the example below:

```
CURSOR
   SELECT
       "DATETIME$1"
   FROM
       "JOURNAL_EXT"
   ORDER BY "JOURNAL_EXT"."DATETIME" ASC
LOOP
  IF "ACTION" = 'I' THEN
      INSERT this row
```

```
  ELSE
   IF "ACTION" = 'U' THEN
      UPDATE this row
  ELSE
        BEGIN
          DELETE FROM
                "TARGET_FOR_JOURNAL_EXT"
END LOOP;
```

This ensures that all consecutive actions are implemented in sequential order and the data is loaded in the order in which the transacation was recorded.

Case Study

This case study shows you how to create ETL logic to load transaction data in a particular order using Warehouse Builder.

Step 1: Import and Sample the Source Flat File, *record.csv* In this example, the flat file *record_csv* stores all transaction records and a timestamp. Import this flat file from your source system using the Warehouse Builder Import Metadata Wizard. Proceed to define the metadata for the flat file in Warehouse Builder using the Flat File Sample Wizard.

Note: You can replace this flat file with a regular table if your system is sourced from a table. In this case, skip to Step 3.

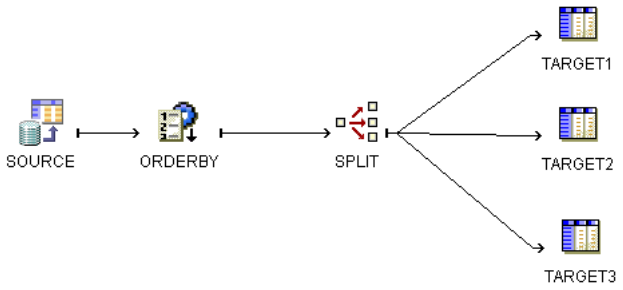
Step 2: Create an External Table To simplify using the sampled flat file object in a mapping in Warehouse Builder, create an external table (JOURNAL_EXT) using the External Table wizard, based on the flat file imported and sampled in Step 1.

The advantage of using an external table instead of a flat file is that it provides you direct SQL access to the data in your flat file. Hence, there is no need to stage the data.

Step 3: Design the Mapping

In this mapping, you move the transaction data from an external source, through an operator that orders the data, followed by an operator that conditionally splits the data before loading it into the target table.

Figure 5–1 ETL Design



The sorter operator enables you to order the data and process the transactions in the exact order in which they were recorded at source. The splitter operator enables you to conditionally handle all the inserts, updates, and deletes recorded in the source data by defining a split condition that acts as the if-then-else constraint in the generated code. The data is conditionally split and loaded into the target table. In this mapping, the same target table is used three times to demonstrate this conditional loading. The

mapping tables TARGET 1, TARGET 2, and TARGET 3 are all bound to the same repository table TARGET. All the data goes into a single target table.

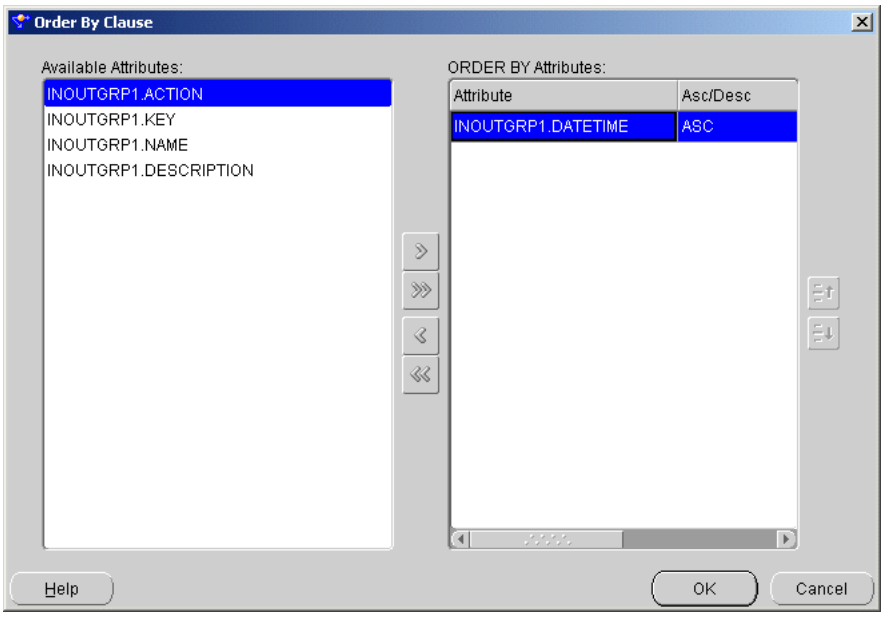
The following steps show you how to build this mapping.

Step 4: Create the Mapping Create a mapping called LOAD_JOURNAL_EXT using the New Mapping Wizard. Warehouse Builder then opens the Mapping Editor where you can build your mapping.

Step 5: Add an External Table Operator Drag and drop a mapping external table operator onto the mapping editor and bind it to the external table JOURNAL_EXT.

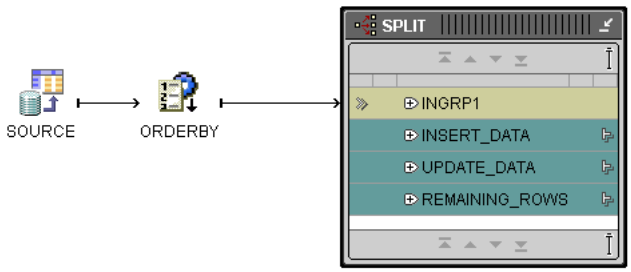
Step 6: Order the Data Add the Sorter operator to define an order-by clause that specifies the order in which the transaction data must be loaded into the target. Figure 5–2 shows you how to order the table based on the timestamp of the transaction data in ascending order.

Figure 5–2 Order By Clause Dialog



Step 7: Define a Split Condition Add the Splitter operator to conditionally split the inserts, updates, and deletes stored in the transaction data. This split condition acts as the if-then-else constraint in the generated code.

Figure 5–3 Adding the Splitter Operator



Define the split condition for each type of transaction. For outgroup INSERT_DATA define the split condition as `INGRP1.ACTION = 'I'`. For UPDATE_DATA, define

the split condition as `INGRP1.ACTION = 'U'`. In Warehouse Builder, the splitter operator contains a default group called REMAINING_ROWS that automatically handles all Delete ('D') records.

Step 8: Define the Target Tables Use the same repository target table three times for each type of transaction: one for INSERT_DATA, one for UPDATE_DATA, and one for the REMAINING_ROWS.

Step 9: The Solution: Configure the Mapping LOAD_JOURNAL_EXT After you define the mapping, you need to configure the mapping to generate code. Because the objective of this example is to process the data strictly in the order it was stored, you must select row based as the default operating mode. In this mode, the data is processed row by row and the insert, update, and delete actions on the target tables take place in the exact order in which the transaction was recorded at source.

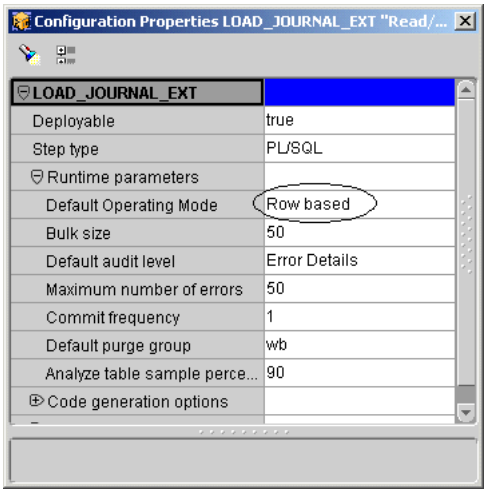
Do not select set-based mode as Warehouse Builder then generates code that creates one statement for all insert transactions, one statement for all update transactions, and a third one for all delete operations. The code then calls these procedures one after the other, completing one action completely before following up with the next action. For example, it first handles all inserts, then all updates, and then all deletes.

To configure the mapping for loading transaction data:

From the Warehouse Builder navigation tree, right-click the LOAD_JOURNAL_EXT mapping and select **Configure.**

Expand the Runtime Parameters node and locate the Default Operating Mode parameter.

Figure 5–4 Configuration Properties for mapping Load_Journal_Ext



Set this parameter to **Row based** as the default operating mode as shown.

In this example, accept the default value for all other parameters. Validate the mapping before generating the code.

Step 10: Generate Code After you generate a mapping, Warehouse Builder displays the results in the Generation Results dialog.

When you inspect the code, you will see that Warehouse Builder implements all consecutive actions in row based mode. This means that the data is processed row by row and Warehouse Builder evaluates all conditions in sequential order using

if-then-else constructions, as shown on page 5-2. The resulting target table thus maintains the sequential integrity of the transactions recorded at source.

6

How Do I Handle Existing PL/SQL Code with Warehouse Builder?

Scenario

A movie rental company periodically updates the customer rental activity in its CUST_RENTAL_ACTIVITY table, where it stores the rental sales and overdue charges data per customer. This table is used for different mailing campaigns. For example, in their latest mailing campaign, customers with high overdue charges are offered the company's new pay-per-view service.

Currently, the movie rental company uses a PL/SQL package to consolidate their data. The existing PL/SQL package needs to maintained manually by accessing the database. This code runs on an Oracle8*i* Database.

```
CREATE OR REPLACE PACKAGE RENTAL_ACTIVITY AS

  PROCEDURE REFRESH_ACTIVITY(SNAPSHOT_START_DATE IN DATE);

END RENTAL_ACTIVITY;
/

CREATE OR REPLACE PACKAGE BODY RENTAL_ACTIVITY AS

 PROCEDURE REFRESH_ACTIVITY(SNAPSHOT_START_DATE IN DATE) IS

   CURSOR C_ACTIVITY IS
    SELECT
      CUST.CUSTOMER_NUMBER CUSTOMER_NUMBER,
      CUST.CUSTOMER_FIRST_NAME CUSTOMER_FIRST_NAME,
      CUST.CUSTOMER_LAST_NAME CUSTOMER_LAST_NAME,
      CUST.CUSTOMER_ADDRESS CUSTOMER_ADDRESS,
      CUST.CUSTOMER_CITY CUSTOMER_CITY,
      CUST.CUSTOMER_STATE CUSTOMER_STATE,
      CUST.CUSTOMER_ZIP_CODE CUSTOMER_ZIP_CODE,
      SUM(SALE.RENTAL_SALES) RENTAL_SALES,
      SUN(SALE.OVERDUE_FEES) OVERDUE_FEES
    FROM  CUSTOMER CUST, MOVIE_RENTAL_RECORD SALE
    WHERE SALE.CUSTOMER_NUMBER = CUST.CUSTOMER_NUMBER AND
          SALE.RENTAL_RECORD_DATE >= SNAPSHOT_START_DATE
    GROUP BY
    CUST.CUSTOMER_NUMBER,
    CUST.CUSTOMER_FIRST_NAME,
    CUST.CUSTOMER_LAST_NAME,
    CUST.CUSTOMER_ADDRESS,
    CUST.CUSTOMER_CITY,
    CUST.CUSTOMER_STATE,
```

```
     CUST.CUSTOMER_ZIP_CODE;

   V_CUSTOMER_NUMBER NUMBER;
   V_CUSTOMER_FIRST_NAME VARCHAR2(20);
   V_CUSTOMER_LAST_NAME VARCHAR2(20);
   V_CUSTOMER_ADDRESS VARCHAR(50);
   V_CUSTOMER_CITY VARCHAR2(20);
   V_CUSTOMER_STATE VARCHAR2(20);
   V_CUSTOMER_ZIP_CODE VARCHAR(10);
   V_RENTAL_SALES NUMBER;
   V_OVERDUE_FEES NUMBER;

BEGIN
   OPEN C_ACTIVITY;
   LOOP
    EXIT WHEN C_ACTIVITY%NOTFOUND;
    FETCH
     C_ACTIVITY
    INTO
     V_CUSTOMER_NUMBER,
     V_CUSTOMER_FIRST_NAME,
     V_CUSTOMER_LAST_NAME,
     V_CUSTOMER_ADDRESS,
     V_CUSTOMER_CITY,
     V_CUSTOMER_STATE,
     V_CUSTOMER_ZIP_CODE,
     V_RENTAL_SALES,
     V_OVERDUE_FEES;

    UPDATE CUST_ACTIVITY_SNAPSHOT
    SET
     CUSTOMER_FIRST_NAME := V_CUSTOMER_FIRST_NAME,
     CUSTOMER_LAST_NAME := V_CUSTOMER_LAST_NAME,
     CUSTOMER_ADDRESS := V_CUSTOMER_ADDRESS,
     CUSTOMER_CITY := V_CUSTOMER_CITY,
     CUSTOMER_STATE := V_CUSTOMER_STATE,
     CUSTOMER_ZIP_CODE := V_CUSTOMER_ZIP_CODE,
     RENTAL_SALES := V_RENTAL_SALES,
     OVERDUE_FEES := V_OVERDUE_FEES,
     STATUS_UPDATE_DATE := SYSDATE
    WHERE
     CUSTOMER_NUMBER = V_CUSTOMER_NUMBER;

    IF SQL%NOTFOUND THEN
       INSERT INTO CUST_ACTIVITY_SNAPSHOT
       ( CUSTOMER_NUMBER,
         CUSTOMER_FIRST_NAME,
         CUSTOMER_LAST_NAME,
         CUSTOMER_ADDRESS,
         CUSTOMER_CITY,
         CUSTOMER_STATE,
         CUSTOMER_ZIP_CODE,
         RENTAL_SALES,
         OVERDUE_FEES,
         STATUS_UPDATE_DATE )
       VALUES
       ( V_CUSTOMER_NUMBER,
         V_CUSTOMER_FIRST_NAME,
         V_CUSTOMER_LAST_NAME,
         V_CUSTOMER_ADDRESS,
```

```
        V_CUSTOMER_CITY,
        V_CUSTOMER_STATE,
        V_CUSTOMER_ZIP_CODE,
        V_RENTAL_SALES,
        V_OVERDUE_FEES,
        SYSDATE )
    END IF;

  END LOOP;


END RENTAL_ACTIVITY;
/
```

Solution

This case study highlights the benefits of importing an existing custom PL/SQL package into Warehouse Builder and using its functionality to automatically maintain, update, and regenerate the PL/SQL code. Warehouse Builder enables you to automatically take advantage of new database features and upgrades by generating code that is optimized for new database versions. For example, if the customer has a PL/SQL package for Oracle8*i*, then by importing it into Warehouse Builder they can generate code for both Oracle8*i* and Oracle9*i*.

Also, by importing a custom package and recreating its operations through a Warehouse Builder mapping, you can transparently run and monitor the operations. Otherwise, you need to manually access the database to verify and update the code. Warehouse Builder also enables you to perform lineage and impact analysis on all ETL operations while the Runtime Audit Browser monitors the running of the code and logs errors.

Case Study

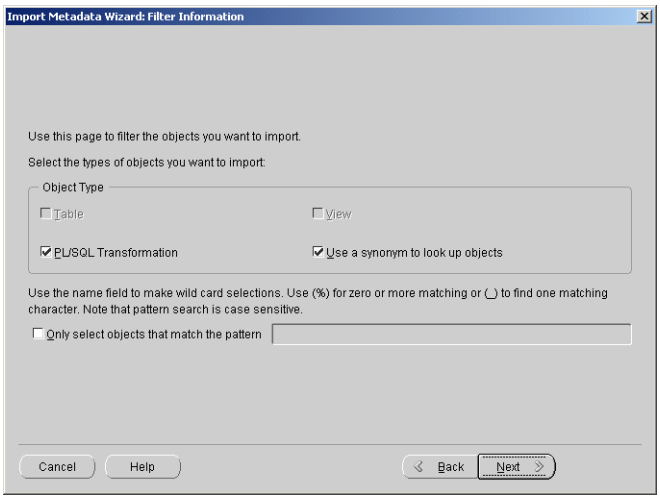
You can migrate the PL/SQL code into Warehouse Builder by taking these steps:

- Import the custom code into Warehouse Builder as a transformation.
- Use the custom transformation in a Warehouse Builder mapping.
- Gradually migrate the legacy PL/SQL code functionality into a new Warehouse Builder mapping and phase out the custom package.
- Generate Code for Oracle 9*i*.

Follow these steps to handle a custom PL/SQL package in Warehouse Builder.

Step 1 Import the Custom PL/SQL Package Expand the Public Transformations node on the Warehouse Builder console to import the PL/SQL package, `refresh_activity(`*`DATE`*`)`, into the Custom folder using the Import Metadata Wizard. On the Filter Information page of this wizard, indicate that you are importing a PL/SQL Transformation as shown in Figure 6–1.

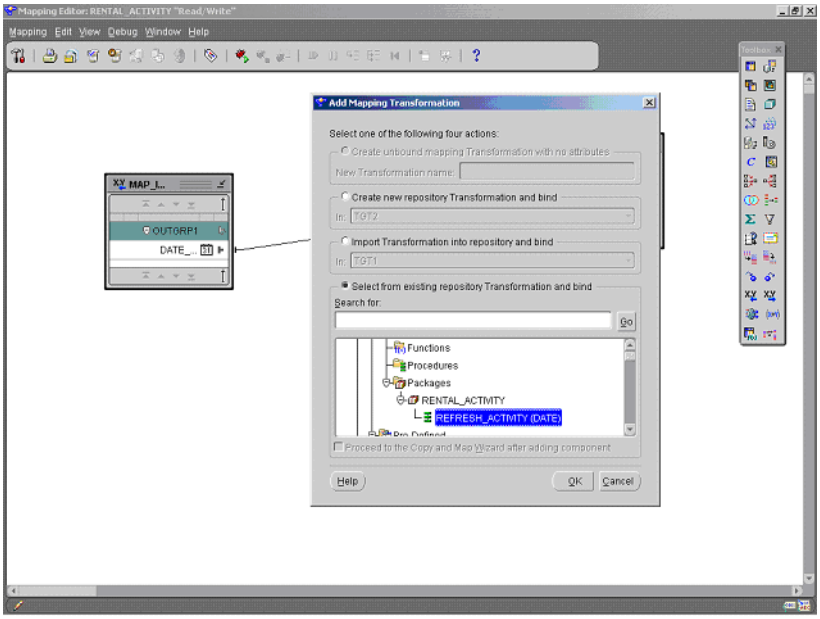
Figure 6–1 Import Metadata WizardL Filter Information Page


Import Metadata Wizard: Filter Information
Use this page to filter the objects you want to import.
Select the types of objects you want to import:
Object Type
Table
View
PL/SQL Transformation
Use a synonym to look up objects
Use the name field to make wild card selections. Use (%) for zero or more matching or (_) to find one matching character. Note that pattern search is case sensitive.
Only select objects that match the pattern
Cancel
Help
Back
Next

After you finish the import, the package `refresh_activity(`*`DATE`*`)` appears under the Packages node of the Custom folder.

Step 2 Create a 'Black Box' Mapping You can use the `refresh_activity(`*`DATE`*`)` procedure directly in a mapping without making any changes to it. In the mapping shown in Figure 6–2, you add a Post-Mapping operator to the mapping, with the package `refresh_activity(`*`DATE`*`)` selected.

Figure 6–2 Using Imported Custom PL/SQL in a Mapping



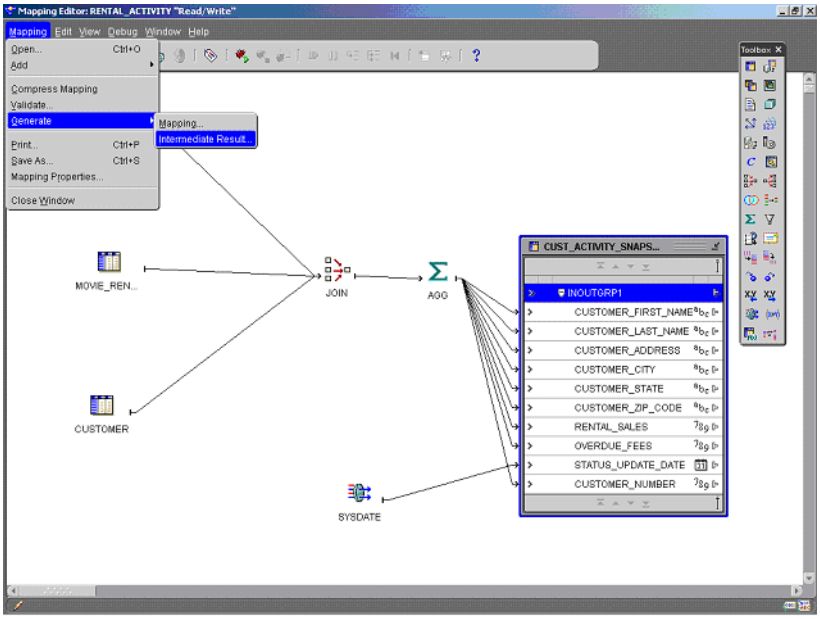
In this example, you can immediately take advantage of the existing custom code. The learning curve and investment on resources is minimal. You may decide to maintain all the existing and developed PL/SQL code in this manner, using Warehouse Builder only to develop new processing units. Warehouse Builder enables you to use mappings that use the legacy code along with the new mappings you create. In such a case, although you can generate code for these mappings in Warehouse Builder, they cannot use Warehouse Builder features to maintain, update, or audit the code.

Because the legacy code is used as a 'black box' that is not transparent to Warehouse Builder, you still need to maintain the legacy code manually. Thus, you cannot take advantage of the Warehouse Builder features, such as runtime audit browser, lineage and impact analysis, and optimized code generation, that rely on infrastructure code and metadata available for Warehouse Builder generated mappings.

Follow the next steps to take advantage of these features in Warehouse Builder and to automatically maintain, monitor, and generate your PL/SQL code.

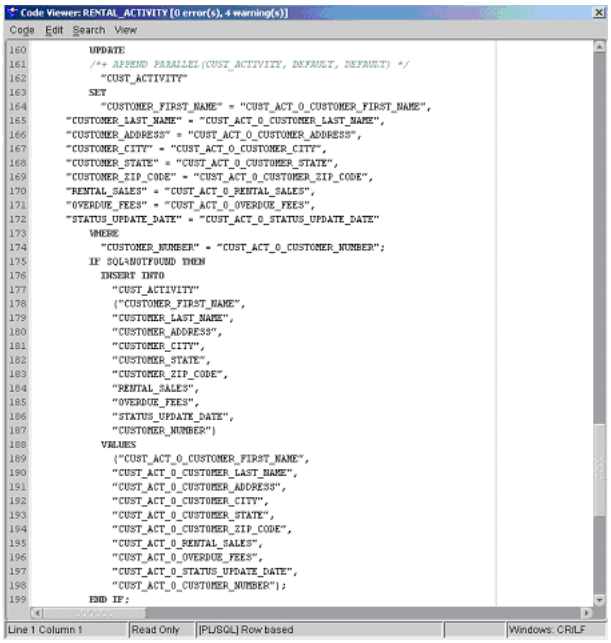
Step 3 Migrate Custom Code into a Mapping To take advantage of the code generation, maintainence, and auditing features available in Warehouse Builder, you can gradually migrate the legacy PL/SQL code functionality into a mapping and phase out the custom 'black box' package. Figure 6–3 shows how the operations performed by the 'black box' mapping, created in step 2, are recreated in the `Rental_Activity` mapping using Warehouse Builder operators and transformations.

Figure 6–3 New Warehouse Builder Mapping



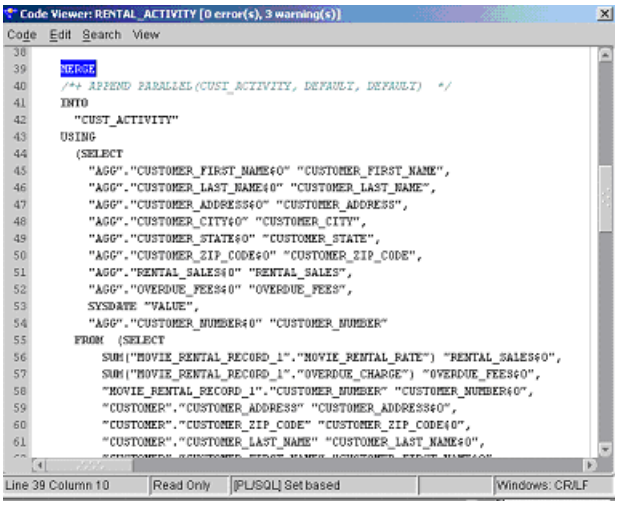
The recommended method is to test out this new mapping by running it side by side with the 'black box' mapping. If the testing is successful and the new mapping can perform all the operations included in the custom code, the 'black box' mappings can be phased out. Warehouse Builder enables you to maintain, update, and generate code from a mapping without perfoming manual updates in the database. Figure 6–4 shows a sample of code generated from the `Rental_Activity` mapping that replicates the operations of the custom PL/SQL package for the movie rental company.

Figure 6–4 Sample Code



Step 4 Generate Code for Oracle9i If you upgrade to Oracle9i version of the database, you only need to re-deploy the `Rental_Activity` mapping created in Step 3. Warehouse Builder generates code optimized for the new database version. Figure 6–5 shows the `MERGE` statement from a sample of code generated for the same mapping for Oracle9*i*.

Figure 6–5 Sample Code for Oracle 9i



No manual steps are required to maintain and generate the new code. Also, you can transparently monitor and maintain their ETL operations. Warehouse Builder enables them to perform lineage and impact analysis on their mappings and the Runtime Audit Browser enables them to track and log errors when running the mappings.

7

How Do I Handle Flat Files with Variable Names?

Scenario

Your company relies on a legacy system that writes data to a flat file on a daily basis and assigns a unique name to the file based on the date and time of its creation. You would like to create a mapping that uses the generated flat files as a source and transforms and loads the data to a relational database. However, mappings require files to have permanent names and, in this situation, the name of the source file changes each time the file is created.

Solution

In Warehouse Builder, you can design a process flow that locates the generated file in a specific directory, renames it to a permanent name you designate, and launches a dependent mapping. You can now use the permanent flat file name as the source for your mapping.

Case Study

This case study describes how to create a process flow and a mapping to extract data from a legacy system that generates flat files with variable names. The process flow relies on the use of an external process activity. Assume the following information for the purposes of this case study:

- **Generated Flat File:** The legacy system generates a flat file containing sales data on a daily basis. It saves the file to the `c:\staging_files` directory and names the file based on the time and date, such as `sales010520041154.dat`. Every generated file is saved to the same directory and begins with the word `sales`, followed by the timestamp information.
- **Permanent Flat File Name:** You decide to rename the generated file name to `s_data.dat`. This is the name you reference as the flat file source in the mapping.
- **Process Activity:** You design a process flow named `OWF_EXT` to execute batch commands in DOS that copies the generated file, saves it as `s_data.dat`, and deletes the originally generated file.

Your objective is to create logic that ensures the generated flat file is renamed appropriately before it triggers the execution of a mapping.

To extract data from a generated flat file with name that varies with each generation, refer to the following sections:

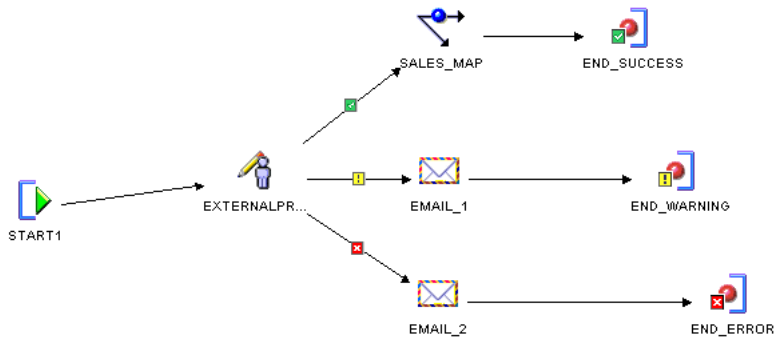
1. "Creating the Process Flow" on page 7-2

2. "Setting Parameters for the External Process Activity" on page 7-2
3. "Configuring the External Process Activity" on page 7-4
4. "Designing the Mapping" on page 7-5
5. "Deploying and Executing" on page 7-5

Creating the Process Flow

Create a process flow that launches a mapping on the condition that the external process activity completes successfully. Your process flow should resemble Figure 7–1.

Figure 7–1 Process Flow with External Process Transitioning to a Mapping



Setting Parameters for the External Process Activity

This section describes how to specify the DOS commands for renaming the generated file. The DOS commands you issue from the external process activity should be similar to the following:

```
copy c:\staging_files\sales*.* c:\staging_files\s_data.dat
del c:\staging_files\sales*.*
```

The first command copies the temporary file into a file with a fixed name `s_data.dat`. The second command deletes the originally generated file.

You can either direct Warehouse Builder to a file containing the script of commands or you can store the commands in the Warehouse Builder user interface. Choose one of the following methods:

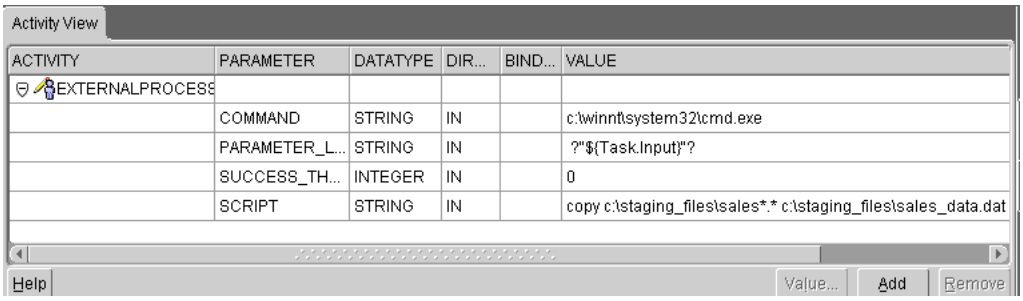
- Method 1: Write a script within Warehouse Builder
- Method 2: Call a script maintained outside of Warehouse Builder

Method 1: Write a script within Warehouse Builder

Choose this method when you want to maintain the script in Warehouse Builder. Consider using this method when the script is small and need not be very flexible.

For this method, write or copy and paste the script into the `VALUE` column of the `SCRIPT` parameter. In the `COMMAND` parameter, type the path to the DOS shell command such as `c:\winnt\system32\cmd.exe`. Also, type the `${Task.Input}` variable into the `VALUE` column of the `PARAMETER_LIST` parameter. Your Activity View should resemble Figure 7–2.

Figure 7–2 External Process Parameters When Script Maintained in Warehouse Builder



Although this case study does not illustrate it, you can use substitution variables in the script when you maintain it in Warehouse Builder. This prevents you from having to update activities when server files, accounts, and passwords change.

Table 7–1 lists the substitute variables you can type for the external process activity. *Working* refers to the machine hosting the Runtime Service, the *local* machine in this case study. Remote refers to a server other than the Runtime Service host. You designate which server is remote and local when you configure the activity as described in "Configuring the External Process Activity" on page 7-4. These values are set when you register the locations at deployment.

Table 7–1 Substitute Variables for the External Process Activity

Variable	Value
${Working.Host}	The host value for the location of the Runtime Service host.
${Working.User}	The user value for the location of the Runtime Service host.
${Working.Password}	The password value for the location of the Runtime Service host.
${Working.RootPath}	The root path value for the location of the Runtime Service host.
${Remote.Host}	The host value for a location other than the Runtime Service host.
${Remote.User}	The user value for a location other than the Runtime Service host.
${Remote.Password}	The password value for a location other than the Runtime Service host.
${Remote.RootPath}	The root path value for a location other than the Runtime Service host.
${Deployment.Location}	The deployment location.

Method 2: Call a script maintained outside of Warehouse Builder

If extra maintenance is not an issue, you can point Warehouse Builder to a file containing a script including the necessary commands. This method is more flexible as it allows you to pass in parameters during execution of the process flow.

The following example shows how to call an external process script outside of Warehouse Builder and illustrates how to pass parameters into the script during execution of the process flow. This example assumes a Windows operating system. For other operating systems, issue the appropriate equivalent commands.

To call a script outside the external process activity:

1. Write the script and save it on the file directory. For example, you can write the following script and save it as `c:\staging_files\rename_file.bat`:

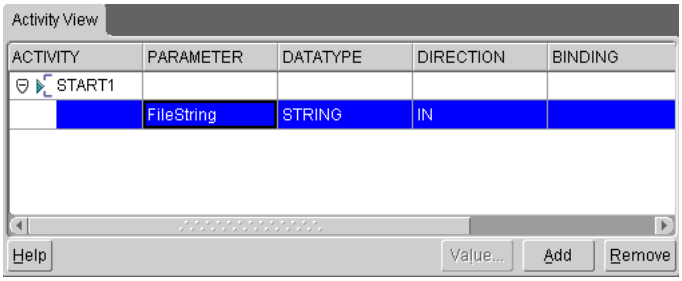
```
copy c:\staging_files\%1*.dat c:\staging_files\s_data.dat
del c:\staging_files\%1*.dat
```

In this sample script, we pass a parameter %1 to the script during the execution of the process flow. This parameter represents a string containing the first characters of the temporary file name, such as `sales010520041154`.

2. Select the start activity on the canvas to view and edit activity parameters in the Activity View displayed in the lower left panel of the Process Flow Editor.

   To add a start parameter, click **Add** on the lower right corner of the Activity View. Create a start parameter named `FILE_STRING` as shown in Figure 7–3. During execution, Warehouse Builder will prompt you to type a value for `FILE_STRING` to pass on to the `%1` parameter in the `rename_file.bat` script.

Figure 7–3 Start Activity in the Activity View



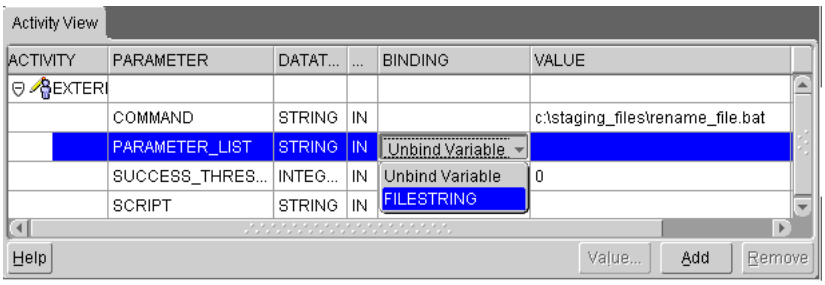
3. Select the external process activity on the canvas and edit its parameters.

   For the `COMMAND` parameter, type the path to the script in the column labelled `VALUE`. If necessary, use the scroll bar to scroll to the right and reveal the column. For this example, type `c:\staging_files\rename_file.bat`.

   For `PARAMETER_LIST`, click the column labelled `BINDING` and select the parameter you defined for the start activity, `FILE_STRING`.

   Accept the defaults for all other parameters for the external process. Your Activity View for the external process activity should resemble Figure 7–4.

Figure 7–4 External Process Parameters When Calling an Outside Script



Configuring the External Process Activity

When you apply conditions to the outgoing transitions of an external process, you must define the meaning of those conditions when you configure the external process activity.

To configure the external process activity:

1. Right-click the process flow on the navigation tree and select **Configure.**

2. Expand the external process activity and the Path Settings. Warehouse Builder displays the configuration settings.
3. Complete this step if you wrote the script in the Warehouse Builder user interface using the substitution variables related to Remote Location, Working Location, and Deployment Location as listed in Table 7–1 on page 7-3. Use the drop down list to select the location for each.

   Because this case study does not use substitution variables, accept the defaults values.
4. Set the Deployed Location to the machine where you deploy the process flow.
5. Set **Use Return as Status** to true.

   This ensures that the process flow uses the external process return codes for determining which outgoing transition to activate. For the process flow in this case study, shown in Figure 7–1 on page 7-2, if the external process returns a success value, the process flow continues down the success transition and executes the downstream mapping.

Designing the Mapping

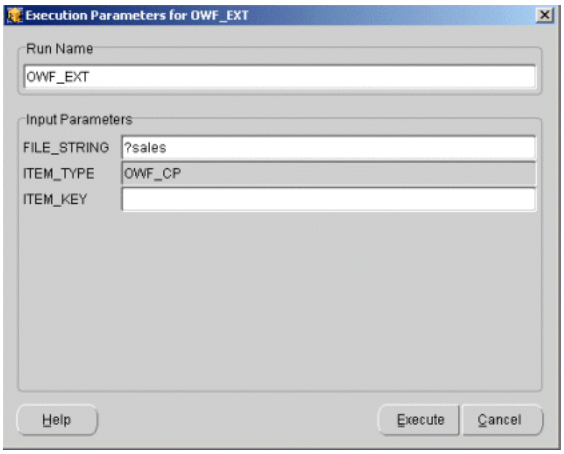
Now you can design a mapping with `s_data.dat` as the source. You can create a PL/SQL mapping or a SQL*Loader mapping. For a PL/SQL, map the flat file source to an external table and design the rest of the mapping with all the operators available for a PL/SQL mapping. For SQL*Loader, map the flat file source to a staging table and limit the mapping to those operators permitted in SQL*Loader mappings.

Deploying and Executing

Deploy the mapping. Also, deploy the process flow package or module containing the process flow `OWF_EXT`.

When you execute the process flow, Warehouse Builder prompts you to type values for the parameter you created to pass into the script, `FILE_STRING`. For this case study, type `?sales` where the question mark is the separator, as shown in Figure 7–5. The external activity then executes the command `rename_file.bat sales`.

Figure 7–5 External Process Activity in the Activity View



8

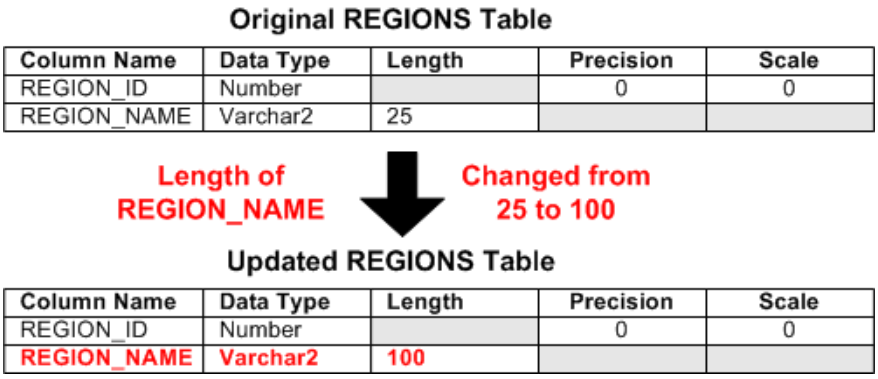
How Do I Update the Target Schema?

Scenario

You are in charge of managing a data warehouse that has been in production for a few months. The data warehouse was originally created using two source schemas, Human Resources (HR) and Order Entry (OE) and was loaded into the Warehouse (WH) target schema. Recently you were made aware of two changes to tables in the HR and OE schemas. The WH schema must be updated to reflect these changes.

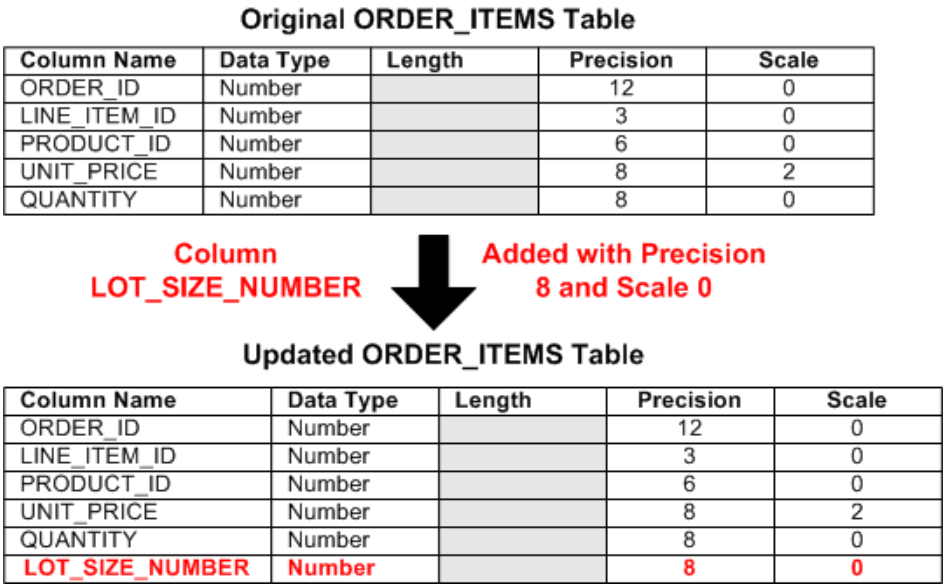
- Change #1: The first change was made to the HR schema as show in Figure 8–1. The length of the *REGION_NAME* column in the *REGIONS* table was extended to 100 characters.

Figure 8–1 Changed REGIONS Table



- Change #2: The second change was made to the OE schema as shown in Figure 8–2. A column called *LOT_SIZE_NUMBER* was added to the *ORDER_ITEMS* table with a precision of 8 and scale of 0.

Figure 8–2 Changed ORDER_ITEMS Table


Original ORDER_ITEMS Table
Column Name | Data Type | Length | Precision | Scale
ORDER_ID | Number | | 12 | 0
LINE_ITEM_ID | Number | | 3 | 0
PRODUCT_ID | Number | | 6 | 0
UNIT_PRICE | Number | | 8 | 2
QUANTITY | Number | | 8 | 0
Column LOT_SIZE_NUMBER
Added with Precision 8 and Scale 0
Updated ORDER_ITEMS Table
Column Name | Data Type | Length | Precision | Scale
ORDER_ID | Number | | 12 | 0
LINE_ITEM_ID | Number | | 3 | 0
PRODUCT_ID | Number | | 6 | 0
UNIT_PRICE | Number | | 8 | 2
QUANTITY | Number | | 8 | 0
LOT_SIZE_NUMBER | Number | | 8 | 0

Solution

In order to update the WH schema, you need to first determine the impact of these changes and then create and execute a plan for updating the target schema. The following steps provide an outline for what you need to do:

- Step 1: Identify Changed Source Objects
- Step 2: Determine the Impact of the Changes
- Step 3: Re-import Changed Objects
- Step 4: Update Objects in the Data Flow
- Step 5: Re-design your Target Schema
- Step 6: Re-Deploy Scripts
- Step 7: Test the New ETL Logic
- Step 8: Update Your Discoverer EUL
- Step 9: Execute the ETL Logic

Case Study

Step 1: Identify Changed Source Objects

The first step in rolling out changes to your data warehouse is to identify changes in source objects. In order to do this, you must have a procedure or system in place that can notify you when changes are made to source objects.

In our scenario, you were made aware by the group managing the HR and OE schemas that some objects had been changed. There were two changes, the first was made to the HR schema. The *REGION_NAME* column was extended from 25 to 100 characters to accommodate longer data. The second change was made to the OE schema. The *LOT_SIZE_NUMBER* column was added and needs to be integrated into the WH schema.

Step 2: Determine the Impact of the Changes

After you have identified the changes, you need to determine their impact on your target schema. For Change #1, made to the HR schema, you need to update any dependent objects. This entails re-importing the *REGIONS* table and then updating any objects that use the *REGION_NAME* column. To identify dependent objects, you can use the Impact Analysis Diagram as shown in Figure 8–3. You also need to update any mappings that use this table.

For Change #2, made to the OE schema, in addition to re-importing the table and updating mappings, you need to find a way to integrate the new column into the WH schema. Since the column was added to keep track of the number of parts or items in one unit of sales, add a measure called *NUMBER_OF_IND_UNITS* to the *SALES* cube in the WH schema and have this measure for each order. Then you need to connect this new column to the *SALES* cube.

Step 3: Re-import Changed Objects

Since two source objects have changed, you must start by re-importing their metadata definitions into your design repository. Select both the *REGIONS* table in the HR schema and the *ORDER_ITEMS* table in the OE schema from the navigation tree and use the Metadata Import Wizard to re-import their definitions.

Warehouse Builder automatically detects that this is an update and proceeds by only updating changed definitions. The Import Results window that displays at the end of the import process displays the details of the synchronization. Click **OK** to continue the import and commit your changes to the repository. If you do not want to continue with the import, click **Undo.**

Step 4: Update Objects in the Data Flow

If the change in the source object altered only existing objects and attributes, such as Change #1 in the HR schema, use Impact Analysis diagrams to identify objects that need to be reconciled.

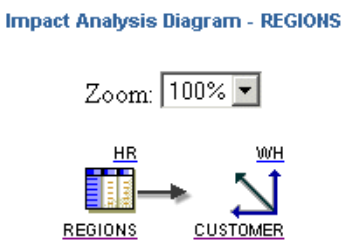
In our scenario, we need to reconcile the column length in all objects that depend on the *REGIONS* table to ensure that the data continues to load properly.

To update objects in the data flow:

1. Select the *REGIONS* table in the HR schema from the navigation tree. Select **View** and then **Impact Analysis.**

   The Design Browser opens as shown in Figure 8–3 and the Impact Analysis diagram reveals that the *CUSTOMER* dimension in the WH schema is the only object impacted by the *REGIONS* table.

Figure 8–3 Impact Analysis Diagram for REGION_NAME



This step requires that you have already set up the Warehouse Builder Design Repository Browser. For more information on setting this up, see the *Oracle Warehouse Builder Installation Guide.*

2. Open the *CUSTOMER* dimension in the Dimension editor and update the Region Name level attribute to 100 character length.

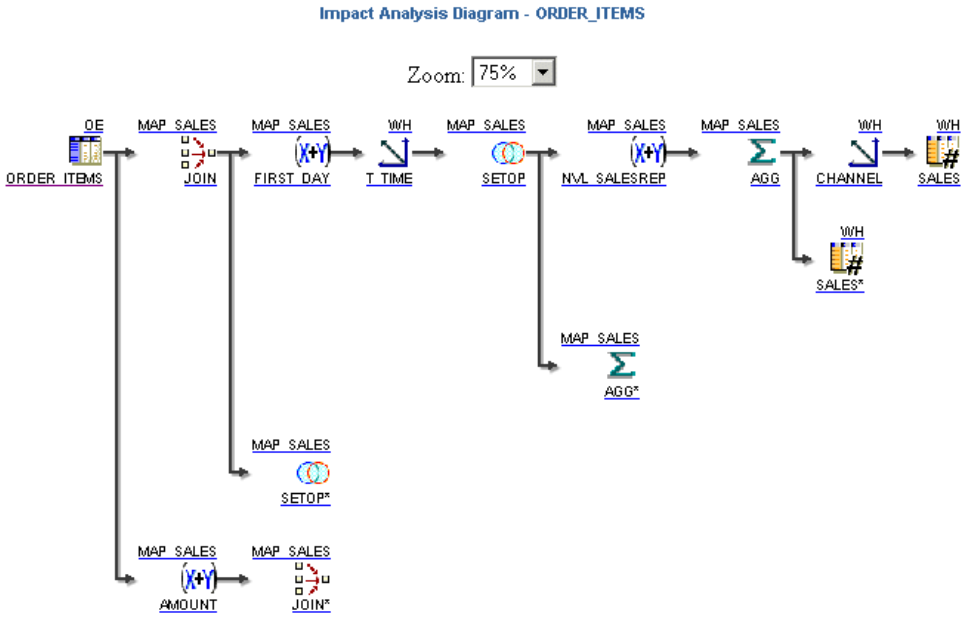
3. Open the *MAP_CUSTOMER* mapping that connects the source to the target and run **Inbound Reconcile** for both the *REGIONS* table operator and the *CUSTOMER* dimension operator.

   The mapping operators must be reconciled to the mapping objects they represent in order to generate code based on the updated objects.

You have now completed updating the metadata associated with Change #1.

For Change #2, since it introduced a new column, you do not need to update the data flow the same way you did for Change #1. Make sure you run **Inbound Reconcile** on all the mappings that use a *ORDER_ITEMS* table operator. From the Impact Analysis Diagram for the *ORDER_ITEMS* table shown in Figure 8–4, we can see that this is only the mapping *MAP_SALES*.

Figure 8–4 Impact Analysis Diagram for ORDER_ITEMS



Step 5: Re-design your Target Schema

Since Change #2 introduced the new *LOT_SIZE_NUMBER* column to the *ORDER_ITEMS* table, you need to re-design your WH target schema to incorporate this new data into your cube. You can do this by adding a new measure called *NUMBER_OF_IND_UNITS* to your *SALES* cube.

To re-design the target schema:

1. Add the measure *NUMBER_OF_IND_UNITS* with the NUMBER data type, precision of 8, and scale of 0 to the *SALES* cube.

2. View the lineage diagram for the *SALES* cube to determine which mappings are contain the *SALES* cube. Perform a reconcile inbound on all *SALES* cube mapping operators.
3. Open the mapping *MAP_SALES* and ensure that the table ORDER_ITEMS is reconciled inbound.
4. Connect the *LOT_SIZE_NUMBER* column in the *ORDER_ITEMS* table to the JOIN, and then to the SETOP, and then add it to the AGG operators. Ensure that you are doing a sum operation in the AGG operator.
5. Finally, connect the *LOT_SIZE_NUMBER* output attribute of the AGG operator to the *NUMBER_OF_IND_UNITS* input attribute of the SALES cube.

Step 6: Re-Deploy Scripts

After the mappings have been debugged, use the Deployment Manager to re-generate and re-deploy scripts. Use the Deployment Manager to discover the default deployment action. Warehouse Builder automatically detects the type of deployment to run.

Step 7: Test the New ETL Logic

After you have reconciled all objects and ensured that the WH target schema has been updated to reflect all changes, test the ETL logic that is be generated from the mappings. Use the Mapping Debugger to complete this task. If you find any errors, resolve them and re-deploy the scripts.

Step 8: Update Your Discoverer EUL

If you are using Discoverer as your reporting tool, proceed by updating your EUL.

To update your Discoverer EUL:

1. Ensure that your SALES collection has been refreshed to include the new measure *NUMBER_OF_IND_UNITS.*
2. From the Project menu, select **Metadata Export** and then **Bridge** to open the Transfer Wizard.
3. Select to transfer to Oracle Discoverer and complete the wizard steps.

   This generates an .EEX file.
4. Import the .EEX file into Discoverer and merge the metadata to complete the EUL update.

Step 9: Execute the ETL Logic

After the mappings have been deployed, execute and load data to the target.

9

How Do I Manage Multiple Versions of my BI Implementation?

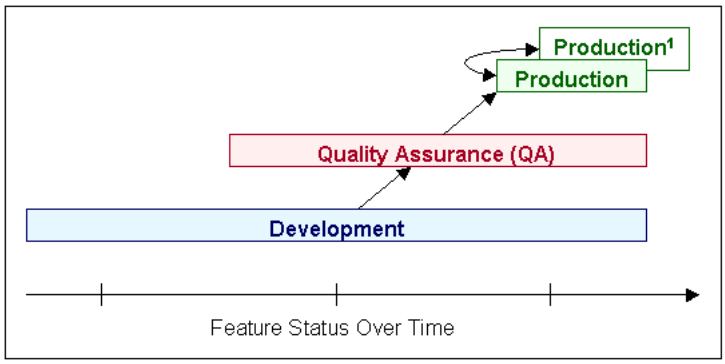
This case study focuses on the life cycle of a business intelligence system. This case study covers two approaches for managing individually changing versions of your BI system once you have implemented in production. The approach you select depends on the phase of your BI system development life cycle.

Scenario

After a period of development and testing, one company implements its BI system in production. The Production version of the system typically changes as new features are incrementally implemented from Development, and as Production bugs are discovered and fixed. At the same time, the Development version of the system continues to evolve with new functionality. This company now has several individually changing versions of the system and faces a challenge familiar to all companies, regardless of how many BI environments they maintain: how to best manage changes in different versions of the system.

One version of this common scenario is depicted in Figure 9–1, where the Development environment is consistently more advanced than the functionality in Production, and QA is somewhere between the two extremes. Development changes are incrementally propagated to QA and subsequently to Production. At the same time, Production has its own cycle of changes, denoted in Figure 9–1 as the shadow environment labeled 'Production^{1}', and used for controlled problem solving. 'Production' and 'Production^{1}' are at the same stage of development, and serve to illustrate the errors that occur in Production, which are fixed and implemented directly in Production, but that must somehow be merged with Development. Other companies may have fewer or more differing environments for their BI systems, but the same maintenance challenges still apply.

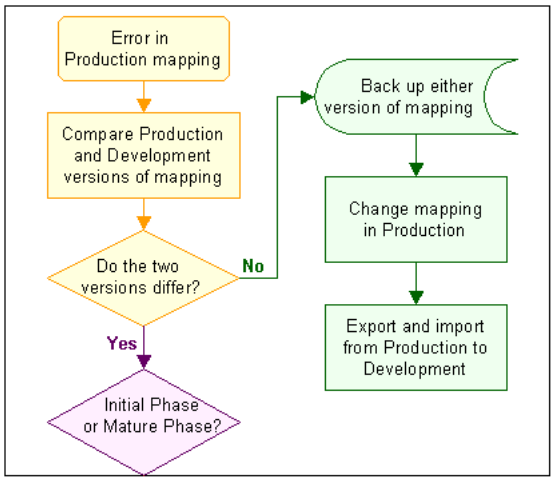
Figure 9–1 Typical Life Cycle of a Business Intelligence System



Companies may need multiple environments for their BI systems, as illustrated in Figure 9–1, because they typically implement incremental changes to the system. However, some companies implement only whole projects in Production. Figure 9–1 does not apply to these companies.

In this case study, a company finds a problem with a mapping in Production. The first step is to compare the Production version of the mapping with the Development version of the mapping, as illustrated in Figure 9–2. If the mapping is identical in both environments, the solution is simple: make the changes in either environment and copy the mapping to override the older version. If the mapping in Production differs from its Development version, then the approach depends on whether the BI system is in its initial or mature phase.

Figure 9–2 Comparing the Production Mapping to Development


Error in Production mapping
Compare Production and Development versions of mapping
Do the two versions differ?
No
Yes
Initial Phase or Mature Phase?
Back up either version of mapping
Change mapping in Production
Export and import from Production to Development

Approach

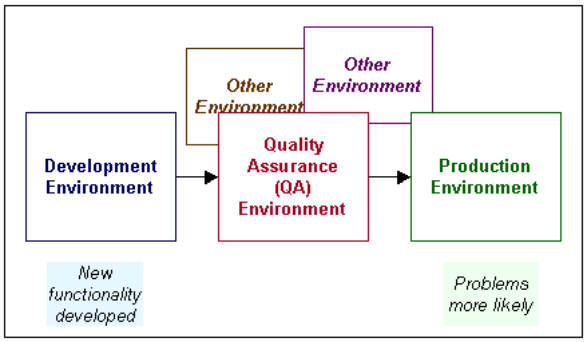
Typically, there are two phases that mark the BI system life cycle: Initial Phase and Mature Phase. The two phases present different needs and call for two different version management methodologies, each of which has benefits and drawbacks.

Initial Phase

After implementation of a business intelligence system in Production, the system is generally in its initial phase, depicted in Figure 9–3. The initial phase is marked by

aggressive changes in the Development environment, coupled with errors sometimes found in Production. Because Production bugs are more likely in this mode, consider a management methodology that facilitates quick updates to each environment.

Figure 9–3 Initial Phase: Changes in Production More Likely



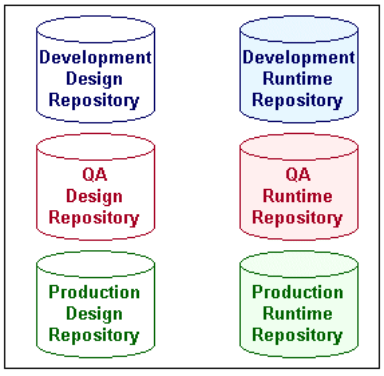
Companies often have two to five different environments. For the initial phase, this company keeps a separate definition of the metadata in each different environment (in this case, Development, QA, and Production). To propagate a change from Production, they export only the portions of the system that have changed and import them into the Development definition.

Case Study

The company has recently implemented its BI system in production, and the system is still in its initial phase, where many additional features are yet to be tested and rolled out. The production system is fairly new, and therefore the occurrence of problems is higher in this phase.

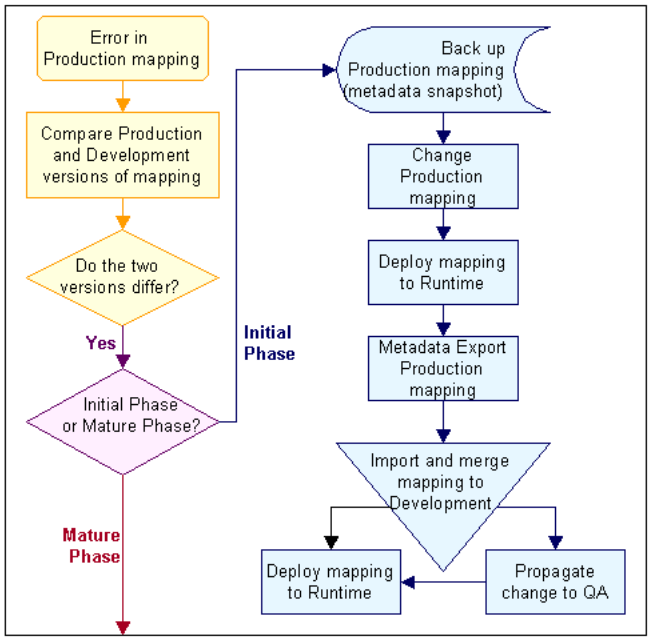
The company decides to keep a separate design repository—or definition of the system design—for each environment, as depicted in Figure 9–4. In addition, they implement their processes into a separate runtime repository for each environment.

Figure 9–4 Initial Phase: Separate Design Repositories



In this example, an error occurs in a Production mapping. The company changes the mapping in Production, then exports its definition, and merges it into Development, as illustrated in Figure 9–5.

Figure 9–5 Initial Phase: Propagate Changes from Production to Development


Error in Production mapping
Compare Production and Development versions of mapping
Do the two versions differ?
Yes
Initial Phase or Mature Phase?
Initial Phase
Mature Phase
Back up Production mapping (metadata snapshot)
Change Production mapping
Deploy mapping to Runtime
Metadata Export Production mapping
Import and merge mapping to Development
Deploy mapping to Runtime
Propagate change to QA

To correct an error found in a Production mapping during the initial phase:

1. For backup, capture the definition of any mapping before modifying it.

   Create a full metadata snapshot of the mapping in the Production Design Repository. Do the same with the Development and QA versions of the mapping. Because you can only restore objects from full snapshots, a full snapshot is essential when you create a backup.

2. Correct the mapping in the Production Design Repository and deploy it to the Production Runtime Repository.

   This results in a changed version of the mapping that must be propagated to other environments.

3. Use Metadata Export to export only the changed mapping from Production.

   Consult the *Oracle Warehouse Builder User's Guide* for instructions on using Metadata Export.

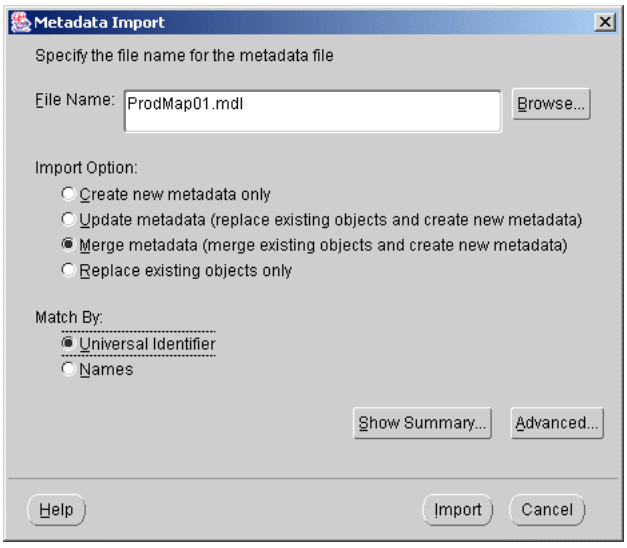
4. Use Metadata Import to import and merge the change to Development and QA, as shown in Figure 9–6.

   - From the Metadata Import dialog Import Options, select **Merge metadata**.
   - From the Metadata Import dialog Match By options, select the **Universal Identifier** option.

     Matching objects by Universal Identifier is important when maintaining multiple individually changing environments.

   Consult the *Oracle Warehouse Builder User's Guide* for more information on using Metadata Import.

Figure 9–6 Metadata Import Options

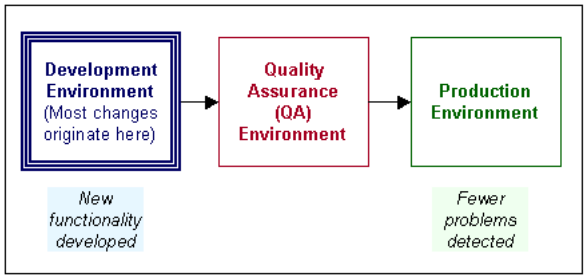


Merging the change into Development and QA can vary in complexity depending on the changed object. If the change in the mapping in this example consists of increasing the column width of a table, the merge is simple. A merge can be more complicated and time-consuming if, for example, join criteria are changed, and other dependencies exist.

Mature Phase

The second is the mature phase, depicted in Figure 9–7. The mature phase is marked by continued changes in the Development environment, but a decrease in changes required in Production.

Figure 9–7 Mature Phase: Fewer Changes in Production



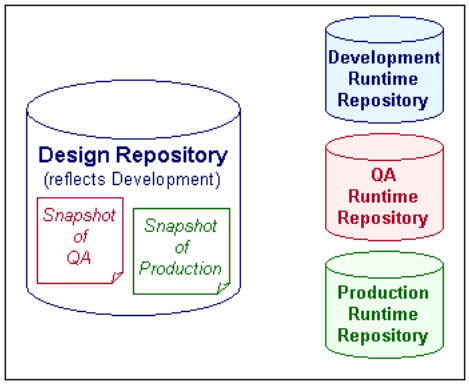
For this mode, the company chooses a methodology that saves space and administration costs: they maintain only one active definition of the BI system design, and this definition reflects the development state of the system. The company stores the design definitions of the QA and Production environments in backup, and extracts and restores changed portions of these systems when required.

Case Study

At this stage, the company's BI system has stabilized and is now in its mature phase. Some additional functionality is still being developed in the Development environment, but fixes originating in Production are rare.

Although they continue to implement their processes into a separate runtime repository for each environment, the company decides to keep only one design repository, as depicted in Figure 9–8.

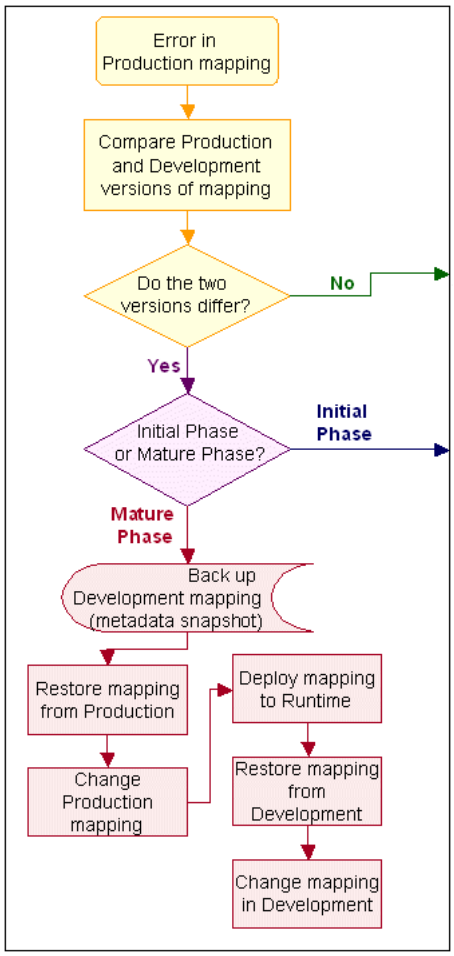
Figure 9–8 Mature Phase: One Design Repository Reflecting Development



The one design repository reflects the Development environment, because it is the one active environment that regularly originates design changes. The design repositories from the QA and Production environments are stored as metadata snapshots inside the Development Design Repository. Snapshots are a backup mechanism that consumes minimal space, and still provides access to any objects that you need to restore. Because design changes rarely originate in Production or QA, storing those definitions in snapshots makes sense.

Although it is more rare during the mature phase, errors still occur in the Production environment. In this example, an error occurs in a Production mapping. The company changes the mapping in Production, then restores its definition from a snapshot in Development and makes the same change there, as illustrated in Figure 9–9.

Figure 9–9 Mature Phase: Propagate Changes from Production to Development



To correct an error found in a Production mapping during the mature phase:

1. Compare the Production version of the mapping in your Production snapshot to the Development version of the same mapping in your Design Repository.

   - If the two differ, the company follows the rest of the steps in this procedure.
   - If the two are identical, they correct the mapping as in Step 8, then deploy it to their Design and Production Runtime Repositories, and then update their Production snapshot with the changed mapping.

   Consult the *Oracle Warehouse Builder User's Guide* for instructions on comparing snapshots to objects, deploying, and on updating snapshots.

2. Back up the Development version of the mapping by creating a full metadata snapshot of it.

   The Development version of the mapping may differ from the Production version if developers have been working on a new iteration of that mapping. This step preserves their work. Creating a full snapshot is essential, because you can only restore from a full snapshot.

3. Restore the mapping in question from the Production snapshot.

   This mapping should be identical to the one running in Production.

Consult the *Oracle Warehouse Builder User's Guide* for instructions on restoring objects from metadata snapshots.

4. Correct the mapping that you have restored from the Production snapshot.
5. Deploy the corrected mapping to the Production Runtime Repository.
6. Remove the existing definition of the mapping from the snapshot of the Production Design Repository and update the snapshot with the new version of the mapping.
7. Restore the mapping from the full snapshot you took as a backup in Step 2.

   This is the mapping from the Development Design Repository. Typically, this mapping has had other work done to it as part of development of new features.

   Optionally repeat this same step for QA.
8. Make the same correction to this Development version of the mapping that you made in Step 4 to the Production version of the mapping.

The cost of this methodology is that every change has to be made at least twice, in the Production and Development versions of the object. The company uses this methodology only because the mature phase does not require frequent changes originating in Production. The benefits of this approach are the minimal administration costs and reduced space requirements on the database.

10

How Do I Inspect Error Logs in Warehouse Builder?

Scenario

While working with Warehouse Builder, the designers need to access log files and check on different types of errors. This case study outlines all the different types of error messages that are logged by Warehouse Builder and how to access them.

Solution

Warehouse Builder logs the following types of errors when you perform different operations:

- Installation Errors
- Metadata Import and Export Errors
- Validation Errors
- Generation Errors
- Deployment and Execution Errors
- Name and Address Server Errors

Case Study

This case study shows you how to retrieve error logs after performing different operations in Warehouse Builder.

Installation Errors

When you run the Oracle Universal Installer to install Warehouse Builder, the installation error logs are automatically stored in:

```
C:\Program
Files\Oracle\Inventory\logs\installActions<timestamp>.log
```

When you run the Warehouse Builder Repository Assistant, the repository installation error logs are stored in:

```
[OWB_ORACLE_HOME]\owb\reposasst\log.txt.0
```

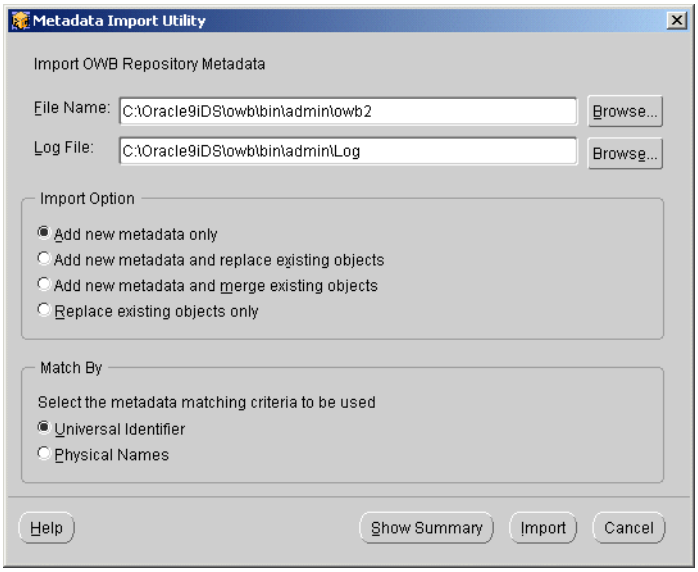
When you run the Warehouse Builder Runtime Assistant, the runtime installation error logs are stored in:

```
[OWB_ORACLE_HOME]\owb\rtasst\log.txt.0
```

Metadata Import and Export Errors

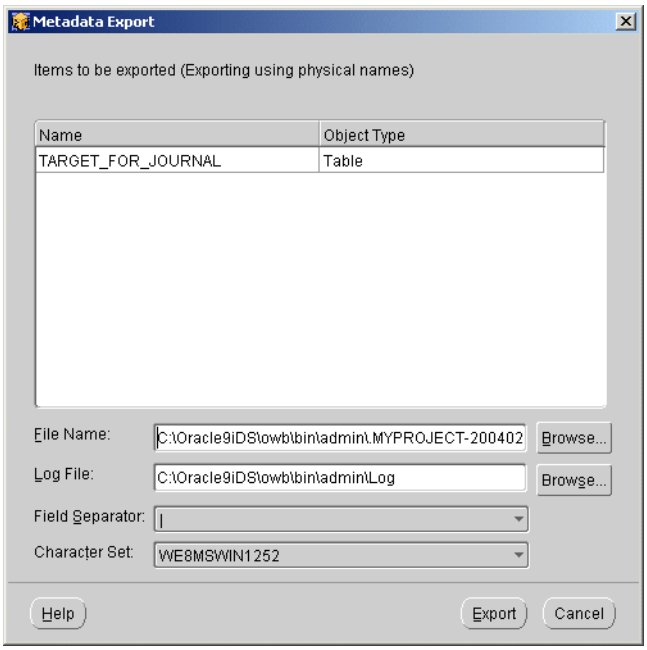
Warehouse Builder Metadata Import: When you import a Warehouse Builder project or specific objects into your repository using the Metadata Import Utility, Warehouse Builder records details of the import process in a log file. You can specify the name and location of this log file from the Metadata Import Utility dialog as shown in Figure 10–1.

Figure 10–1 Log File Location in the Metadata Import Utility Dialog



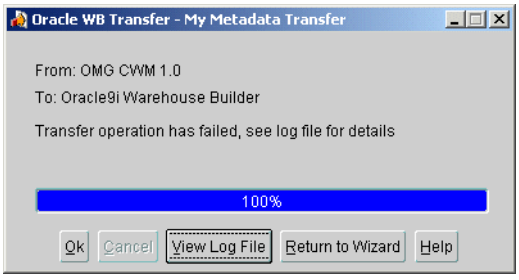
Warehouse Builder Metadata Export: When you export a Warehouse Builder project or specific objects using the Metadata Export Utility, then Warehouse Builder records the details of the export in a log file. You can specify the name and location of this log file from the Metadata Export dialog, as shown in Figure 10–2.

Figure 10–2 Metadata Export Log File Location



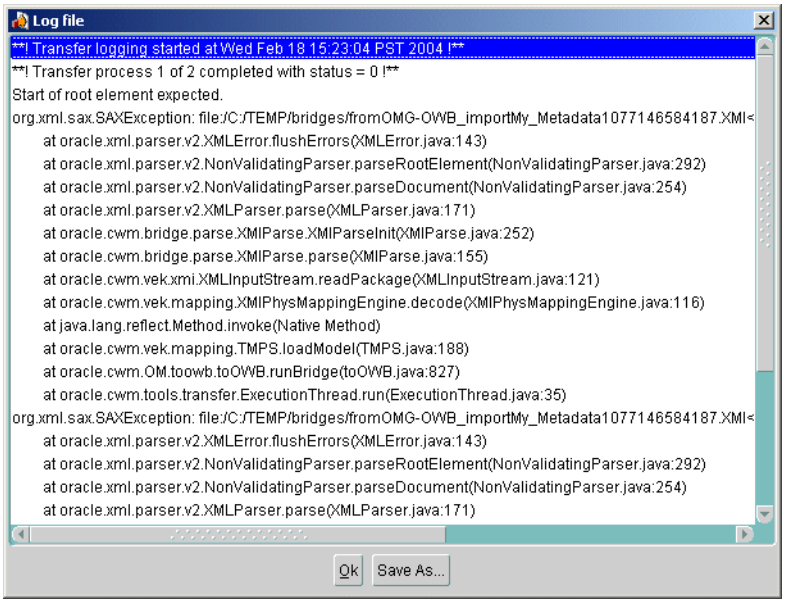
Metadata Import or Export Using the Transfer Wizard: If you are importing or exporting design metadata using the Warehouse Builder Transfer Wizard, then you can view the log file after the import is complete. Warehouse Builder displays the My Metadata Transfer dialog, as shown in Figure 10–3.

Figure 10–3 Transfer Wizard Log File



Click **View Log File** to view the log file, as shown in Figure 10–4. Click **Save As** to save the log file to your local system.

Figure 10–4 Metadata Import Log File

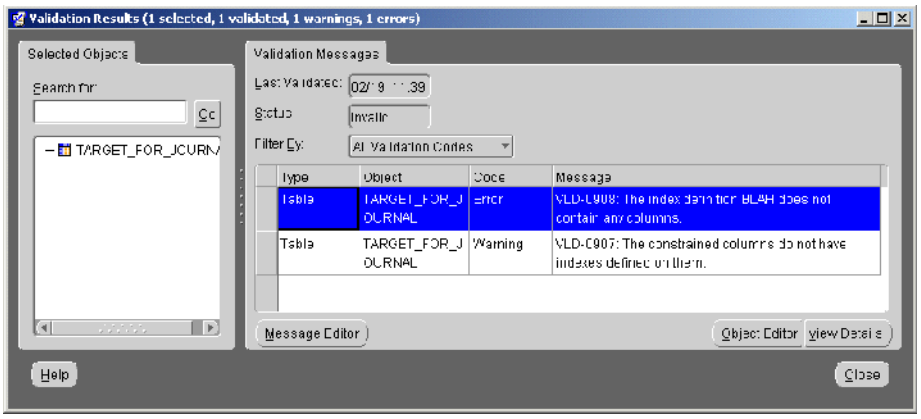


Validation Errors

In Warehouse Builder, you can validate all objects from the console tree by selecting **Object,** then **Validate.** After the validation is complete, the validation messages are displayed in the Validation Results window, as shown in Figure 10–5. Click **Message Editor** to view the errors in a text editor that enables you to edit and save the messages to your local system.

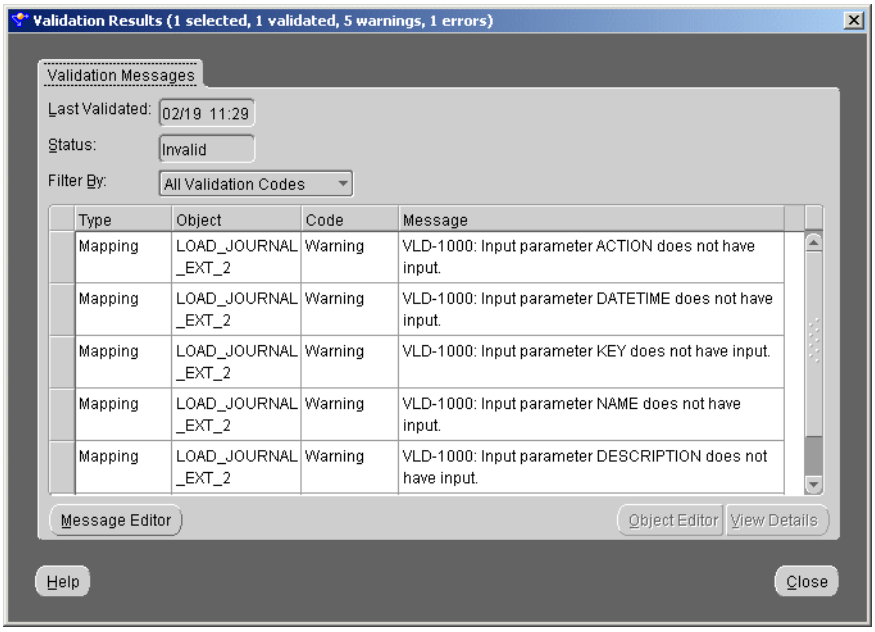
When you select a particular error message, the **Select Editor** and **View Details** buttons are activated. Click **Object Editor** to display the editor for the selected object. For example, in Figure 10–5, if you click **Object Editor,** Warehouse Builder displays the Table Editor to enable you to fix the errors. When you click **View Details,** Warehouse Builder displays the detailed error message.

Figure 10–5 Validation Error Messages



You can also validate mappings from the Mapping Editor by selecting **Mapping,** then **Validate.** The validation messages and errors are displayed in the Validation Results window, as shown in Figure 10–6.

Figure 10–6 Validation Error Messages



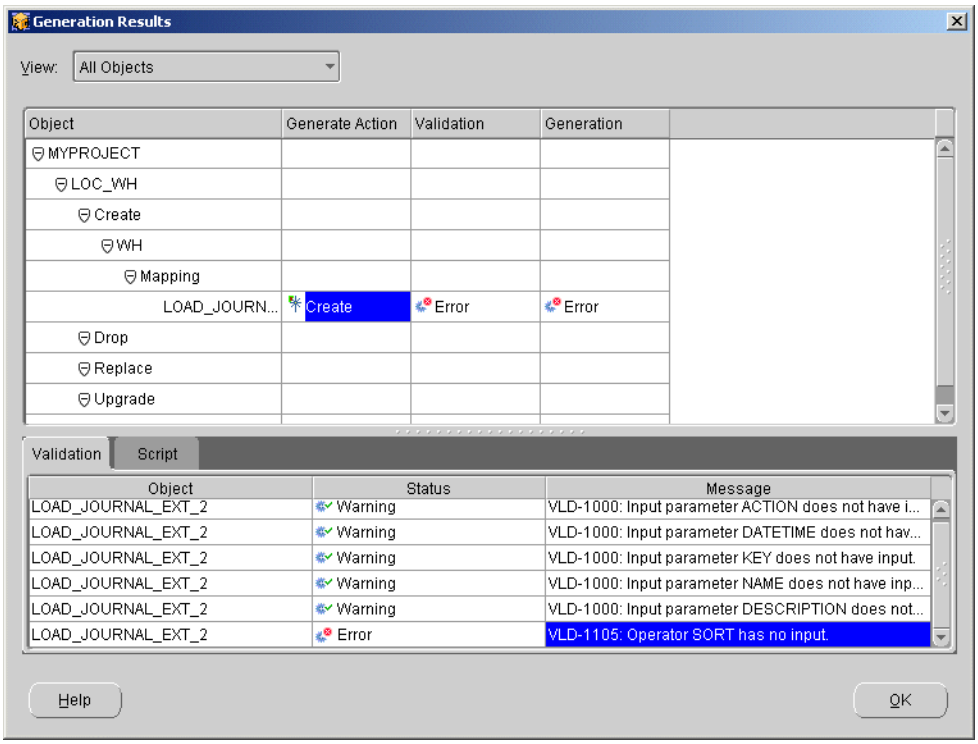
This window is similar to the Validation Results window displayed from the console. You can save the message to your local system by clicking **Message Editor.**

Warehouse Builder saves the last validation messages for each previously validated object. You can access these messages at any time by selecting the object from the console tree and then selecting **View** and then **Validation Messages.** The messages are displayed in the Validation Results window.

Generation Errors

After you generate scripts for Warehouse Builder objects, the Generation Results window displays the generation results and errors, as shown in Figure 10–7. Double-click an error under the messages column to display a message editor that enables you to save the errors to your local system.

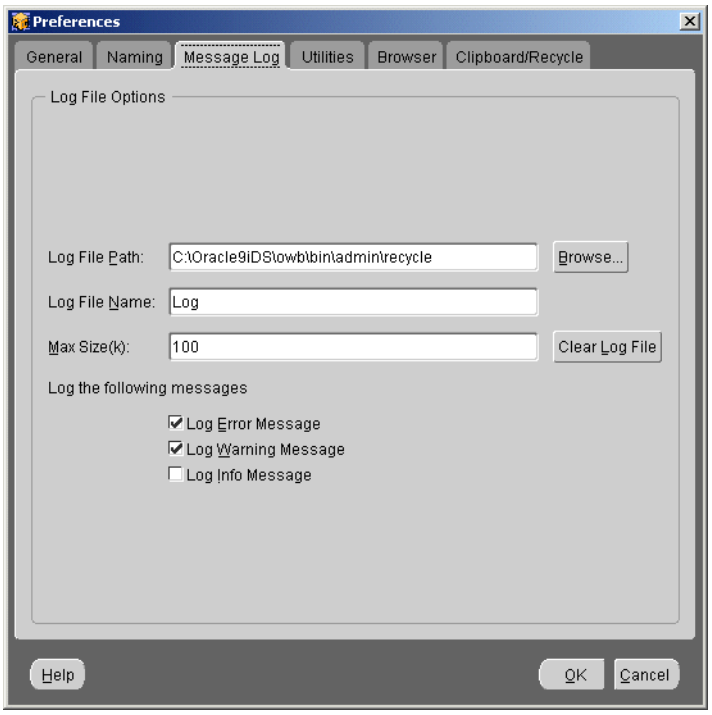
Figure 10–7 Generation Results Window



Deployment and Execution Errors

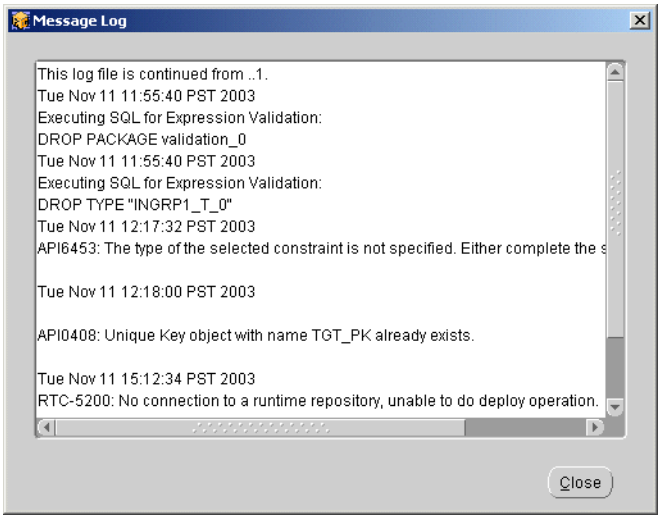
You can store execution or deployment error and warning message logs on your local system by specifying a location for them. From the Warehouse Builder console, select **Project,** then **Preferences.** Click the Message Log tab to set the log file path, name, and maximum size. You can also select the types of logs you want to store.

Figure 10–8 Setting Error Log Preferences



You can view this log of deployment and error messages from the Warehouse Builder console by selecting **View** then **Messages Log.** This Message Log window, as shown in Figure 10–9, is read-only.

Figure 10–9 Message Log Window


Message Log
This log file is continued from ..1.
Tue Nov 11 11:55:40 PST 2003
Executing SQL for Expression Validation:
DROP PACKAGE validation_0
Tue Nov 11 11:55:40 PST 2003
Executing SQL for Expression Validation:
DROP TYPE "INGRP1_T_0"
Tue Nov 11 12:17:32 PST 2003
API6453: The type of the selected constraint is not specified. Either complete the s
Tue Nov 11 12:18:00 PST 2003
API0408: Unique Key object with name TGT_PK already exists.
Tue Nov 11 15:12:34 PST 2003
RTC-5200: No connection to a runtime repository, unable to do deploy operation.
Close

Runtime Audit Browser: If an error occurs while transforming or loading data, the audit routines report the errors into the runtime tables. You can easily access these error reports using the Runtime Audit Browser (RAB). The RAB provides detailed information about past deployments and executions. These reports are generated from data stored in the Runtime Repositories. Click the Execution tab in the Execution reports to view error messages and audit details.

Name and Address Server Errors

If you are using the Name and Address cleansing service provided by Warehouse Builder, you can encounter related errors.

Name and address server start up and execution errors can be located at:

[OWB_ORACLE_HOME]\owb\bin\admin\NASver.log

If your Name and Address server is enabled in:

[OWB_ORACLE_HOME]\owb\bin\admin\NameAddr.properties:TraceLevel=1

,

then it produces the log file NASvrTrace.log.

11

How Do I Export Metadata to Oracle Discoverer?

Scenario

Your company analyzes its sales by these parameters: Products, Channels, Geographies, Promotions, and Time. You have successfully designed and deployed a five dimension star schema in Warehouse Builder to perform this analysis. Now your company wants to use Oracle Discoverer as their data analysis tool. You want to maintain a single dimensional design by exporting the same design metadata from Warehouse Builder to Discoverer.

Solution

Warehouse Builder enables you to export your dimensional design directly to Oracle Discoverer. Although your company uses Warehouse Builder as its design tool and Discoverer as its analytical tool, you do not need to maintain two separate designs.

Case Study

Follow these steps to export your design metadata from Warehouse Builder to Discoverer:

- Create a collection including the dimensional design objects you want to export.
- Export the collection to Discoverer using the Transfer Bridge.
- Import the metadata into Oracle Discoverer.
- View the data and create reports using the Discoverer Desktop edition.

Step 1: Create a Collection in Warehouse Builder

Create a collection that includes all the Warehouse Builder design objects you want to export. In this case, the objects are all the dimensions: `Products`, `Channels`, `Geographies`, `Promotions`, and `Time` and a cube: `Sales`.

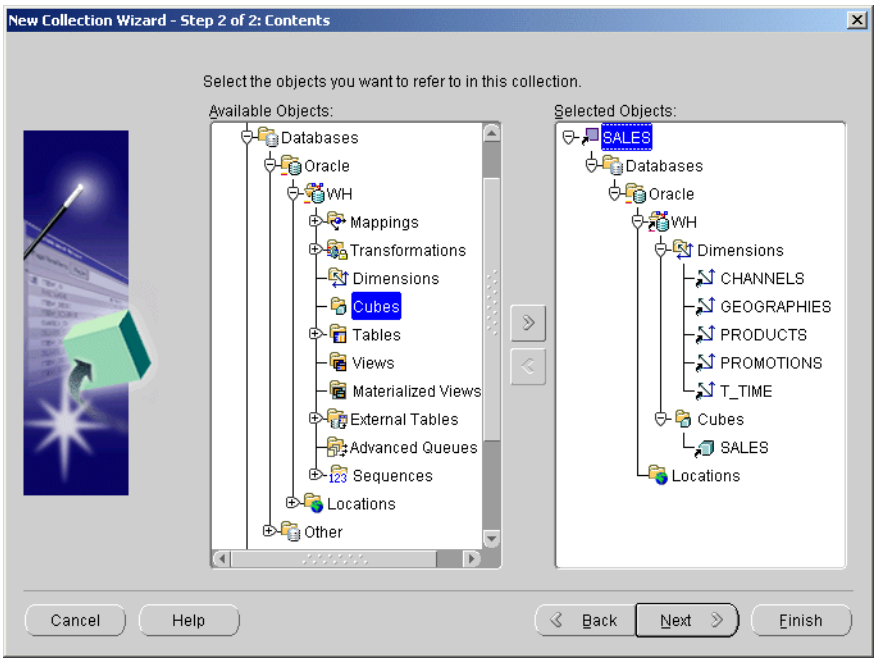
To create a collection:

1. From the Warehouse Builder navigation tree, right-click the Collections node and select **Create Collection.**

   The New Collection Wizard displays.

2. Name the collection. In this scenario, the collection is called `Sales`.
3. From the **Available Objects** field, select all the design objects you want to export. Figure 11–1 shows the selected dimensions and a cube.

Figure 11–1 New Collection Wizard: Selecting Objects to Export



4. Click **Next** to review your selections. Click **Finish** to finish creating the collection.

   This collection is stored under the Collections node on the navigation tree.

Step 2: Export Warehouse Builder Metadata to Discoverer

After you create a collection containing all the design objects you want to export, use the Warehouse Builder Transfer Wizard to perform the export. Follow these steps:

1. From the Warehouse Builder console, select **Project,** then **MetaData Export,** then **Bridge.**

   The Transfer Wizard displays. Click **Next** to go to the Source and Target Metadata Locations page.

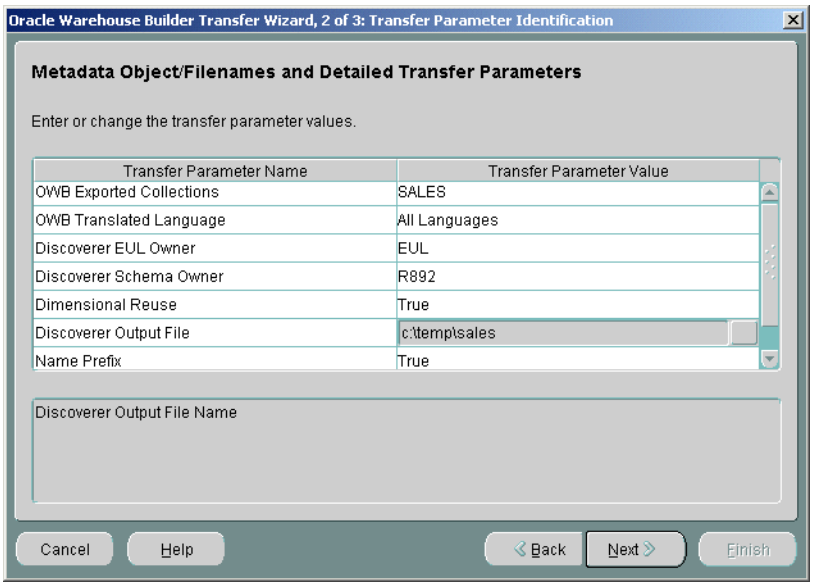
2. In the **To** field, select **Oracle9i Discoverer.**

3. In the Transfer Parameter Identification page, provide the following Transfer Parameter Values as shown in Figure 11–2:

   - OWB Exported Collections: Select the `Sales` collection.
   - Discoverer Schema Owner: Type the owner of the data objects that you want Discoverer to query. This is usually the Warehouse Builder target schema.
   - Dimensional Reuse: Set to True you want to create an additional Discoverer folder for a dimension that is used more than once in a dimensional query. For example, your company uses time twice, as shipping and billing date, in the same query.
   - Discoverer Output File: Indicate the name and location of the file that will contain the metadata exported from Warehouse Builder. For example, `c:\temp\sales.eex`. You will import this file into Discoverer later.
   - Name Prefix: Indicate whether you want to prefix the Discoverer folder items (dimension levels) with the Warehouse Builder level names. This parameter is set to false for this example.

- Accept default values for all other parameters.

Figure 11–2 Warehouse Builder Discoverer Transfer Parameters



4. Click **Next** to review the parameters. Click **Finish** to export the metadata file.

   The output file is stored in the location you specified in the parameter value.

Step 3: Import Warehouse Builder Metadata File into Discoverer

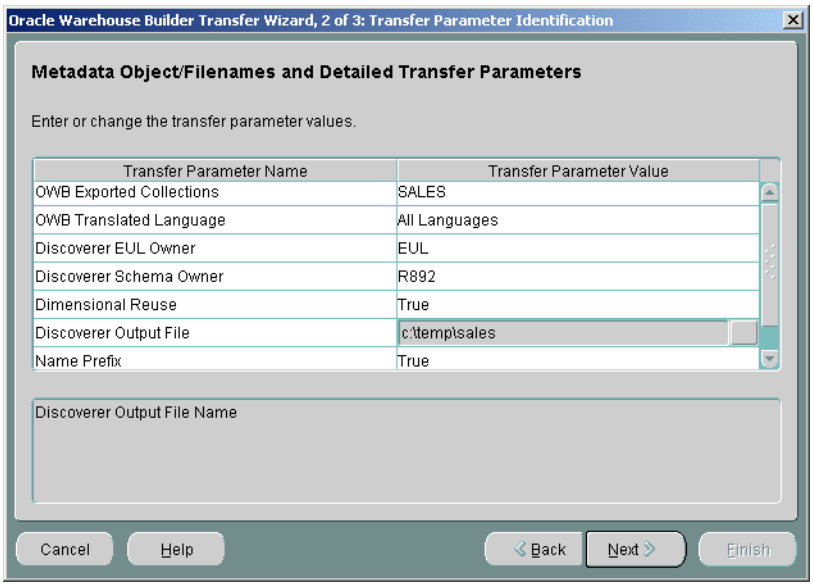
After you finish exporting the Warehouse Builder design to a file, you need to import the file into Discoverer. Follow these steps:

1. Start Discoverer Administrator.
2. After login, the Load Wizard displays. Click **Cancel.**
3. From the **File** menu select **Import.**

   The Import Wizard displays.
4. Click **Add** to select the file `sales.eex` that was generated by Warehouse Builder.
5. Accept the default values in the Import Wizard: Step 2 and click **Next.**
6. On the Import Wizard: Step 3 and click **Start.**

   Discoverer creates a business area with the same name as the collection that you exported from Warehouse Builder. In this example, the collection and business area are called `Sales`.
7. Inspect the imported objects in the business area in Discoverer Administrator as shown in Figure 11–3.

Figure 11–3 Discoverer Administrator Workarea



The metadata import is now complete. Warehouse Builder thus enables you to maintain a single design metadata source that you can periodically import into Discoverer for reporting purposes.

Step 4: View the Data from Discoverer Desktop Edition

After you create a workarea containing Warehouse Builder design objects in Discoverer, you can view the data and run reports on it by creating a workbook. Follow these steps:

1. Start Discoverer Desktop.

   The Workbook Wizard displays.

2. Click **Create an Existing Workbook** and select **Crosstab** as the workbook type.

3. In step 2 of the Workbook Wizard, select the `Sales` business area. Also select the sales measures and dimension attributes that will be used to build this workbook.

   You will notice that the dimension and cube structures from Warehouse Builder have been seamlessly imported into Discoverer.

4. In step 3 of the Workbook Wizard, design the workbook layout by defining the horizontal and vertical axis. Click **Finish.**

   Now you can view the data in the workbook and run analytical reports on it.

12

How Do I Export a Warehouse Builder Design to a File?

Scenario

A company is designing Warehouse Builder mappings in a development and testing environment. When the design work is complete, they transfer the designs to a Warehouse Builder production environment. The company wants to maintain a single design base that can be deployed from different environments. In Warehouse Builder, although it is possible to deploy the same design in two different runtime environments, this is only possible if both environments have identical configurations, which is not most often not the case.

Solution

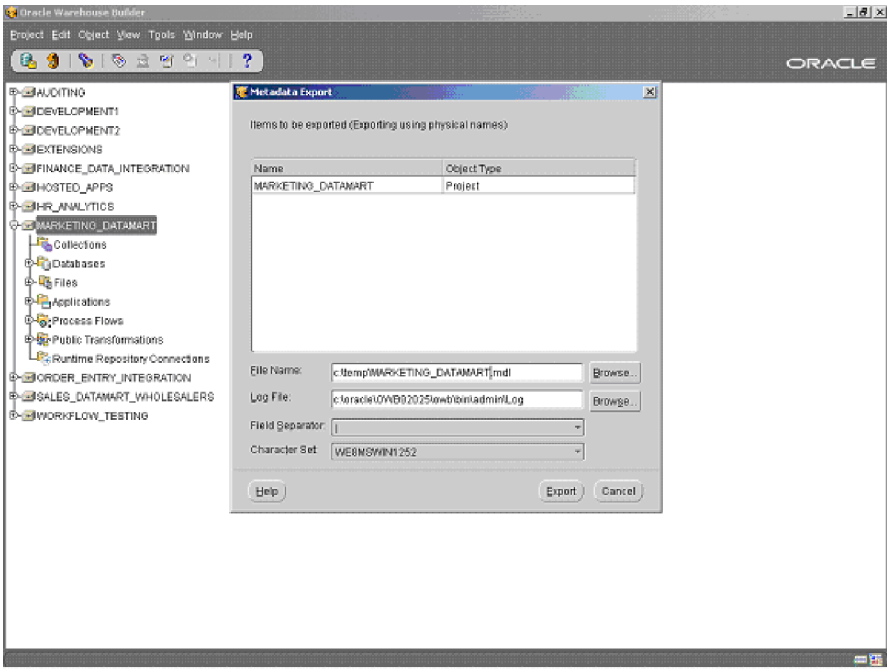
The customer can deploy a single Warehouse Builder design to two completely different production environments in the following way:

- Export the design metadata from the development environment to a .mdl file.
- Import the .mdl file into the production environment.
- Re-configure the imported design for deployment.

Step 1: Export the Design Metadata to an .MDL File

First select the Warehouse Builder design project you want to export, then from the **Project** menu select **MetaData Export** and then **File.** Warehouse Builder displays the Metadata Export dialog as shown in Figure 12–1.

Figure 12–1 Metadata Export Dialog



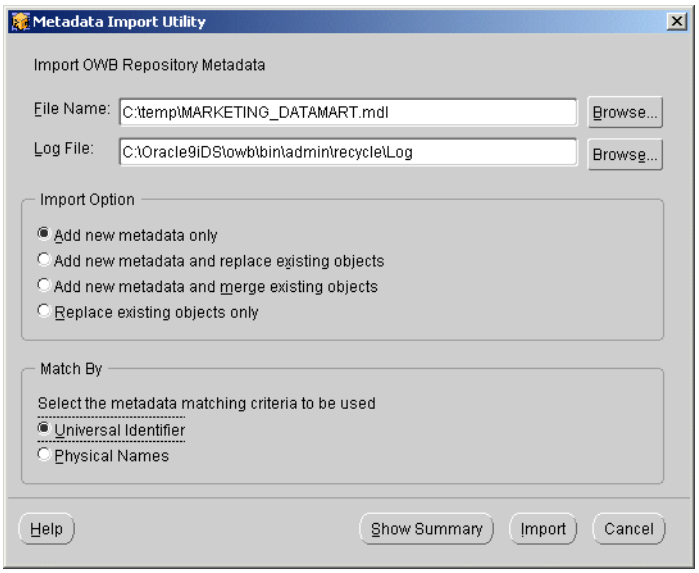
In the Metadata Export dialog, provide a file name and a path to store the metadata on your local system. Also, provide a name for the log file and a location for it on your local system.

Click **Export** to export the design metadata as a .mdl file as indicated in the Metadata Export dialog. In this example, the file is stored in the following format, C:\temp\MARKETING_DATAMART.mdl. After the export is complete, you must inspect the export results. If there are error messages, you can inspect the export log file by clicking **View Log File.**

Step 2: Import Metadata into the Production Environment

Now you can import the .mdl design file exported from the development environment into the production environment. To import a .mdl file, first log into your production repository using your Warehouse Builder client. From the **Project** menu in the console window, select **MetaData Import** and then **File.** Warehouse Builder displays the Metadata Import Utility window as shown in Figure 12–2.

Figure 12–2 Metadata Utility Window



In the File Name field, type or browse for the location of the exported .mdl design file on your local system. In this example, the file was exported to this location, *C:\temp\MARKETING_DATAMART.mdl.* Next, provide a location for the import log file on your local system. Now choose from one of the import options. In this example, we choose **Add new metadata only.** If you are re-importing this file, you can choose from one of the other options as well.

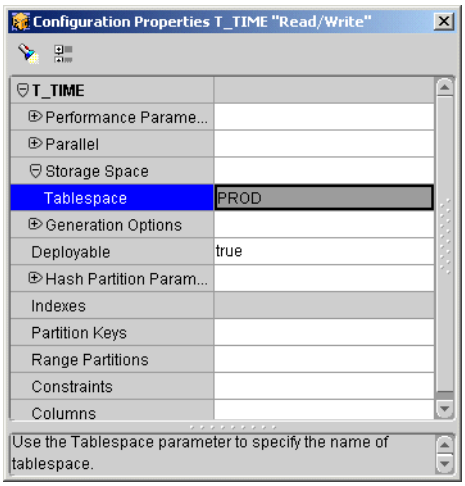
The Import Utility also provides you different Match By options compare imported objects with those existing in the repository. Every object in a Warehouse Builder design repository contains a universal identifier. If you select match by Universal Identifier, then during the import, the objects will be considered the same only if their universal identifier ids are the same. This means only if the objects in the .mdl file and the objects in the repository have matching ids. If you change the match by criteria to Physical Names, then Warehouse Builder matches object names instead of universal identifiers to establish the similarity of any two objects. In this example, we accept the default option of matching by Universal Identifiers.

Click **Import** to import the design file exported from your development environment. Inspect the import results and check the log file in case of any errors.

Step 3: Configure the Imported Metadata for the Production Environment

After you import the design file, you can change the configuration parameters for the design according to the requirements of your production runtime environment. For example, in this company's production environment, the time dimension must be created in a different tablespace than the one configured in the development design. You can change the physical configuration parameters for any object using the Configuration Properties window.

Figure 12–3 Changing Physical Configuration



In this example, you can change the configuration properties for the time dimension by right-clicking the dimension name and selecting **Configure** from the pop-up menu. Now change the Tablespace parameter under the Storage Space node from DEV to PROD, as shown in Figure 12–3. Use this method to change any other parameters for the imported design objects before deploying them.

Warehouse Builder thus enables you to easily deploy the same design in two completely different runtime environments with different design repositories.

Index

B

BI life cycle, 9-1

C

CA ERwin
  importing designs from, 4-1
changes
  propagating through multiple BI environments, 9-1
  rolling out to the target schema, 8-1

D

data analysis
  on metadata designed in Warehouse Builder, 11-1
data model designs
  importing from third-party tools, 4-1
deploying
  deployment errors, 10-1
  single design to different environments, 12-1
designs
  deploying one design to different environments, 12-1
  exporting design metadata from Warehouse Builder to Discoverer, 11-1
  sharing metadata between design and analytical tools, 11-1
Discoverer
  exporting Warehouse Builder metadata to Discoverer, 11-1

E

error logs
  interpreting error logs, 10-1
ERwin
  importing designs from, 4-1
Excel files
  loading data from, 3-1
execution
  errors, 10-1
exporting
  metadata to Discoverer, 11-1
  Warehouse Builder design to a file, 12-1
external tables
  loading data from flat files, 1-1

F

file transfer
  in process flows, 2-1
files
  exporting Warehouse Builder design to a file, 12-1
flat files
  loading data from, 1-1
  variable names, in process flows, 7-1
FTP
  using in process flows, 2-1

G

generation
  errors, 10-1

I

impact analysis
  rolling out changes to target schema, 8-1
implementation
  multiple versions of BI implementation, 9-1
importing
  designs from third-party tools, 4-1
installation
  errors, 10-1

L

loading
  data from Excel files, 3-1
  data from flat files, 1-1
  transaction data, 5-1
logs
  interpreting error logs, 10-1

M

metadata
  exporting to Discoverer, 11-1
  exporting Warehouse Builder design to a

file, 12-1
import and export errors, 10-1
Microsoft Excel
loading data from, 3-1

N

Name and Address server
errors, 10-1
names
flat files with variable names in process flows, 7-1

O

Oracle Discoverer
See Discoverer

P

PL/SQL code
handling existing PL/SQL code, 6-1
process flows
handling flat files with variable names, 7-1
transferring remote files with FTP, 2-1
production
changes in, 9-1

R

remote files
tranferring, 2-1
row-based vs. set-based
loading transaction data, 5-1

S

set-based vs. row-based
loading transaction data, 5-1
SQL*Loader
loading data from flat files, 1-1

T

target schema
rolling out changes, 8-1
transaction data
loading, 5-1
transferring
remote files, 2-1

U

updating
target schema, 8-1

V

validation
errors, 10-1